O_2Web User Manual

O_2C

Release 5.0 - April 1998

Who should read this manual

This manual describes how to develop a World Wide Web server using the O_2 system. It describes how to write in HTML and program an O_2Web server. The manual contains a comprehensive list of O_2Web methods and commands.

Other documents available are outlined, click below.

See O2 Documentation set.

Table of Contents

This manual is divided into the following chapters:

TABLE OF CONTENTS

TABLE OF CONTENTS

1 Introduction

GENERAL OVERVIEW OF O_2WEB

Congratulations! You are now a user of O_2Web. You will find that building your WWW service on top of the O_2 database system has several key benefits.

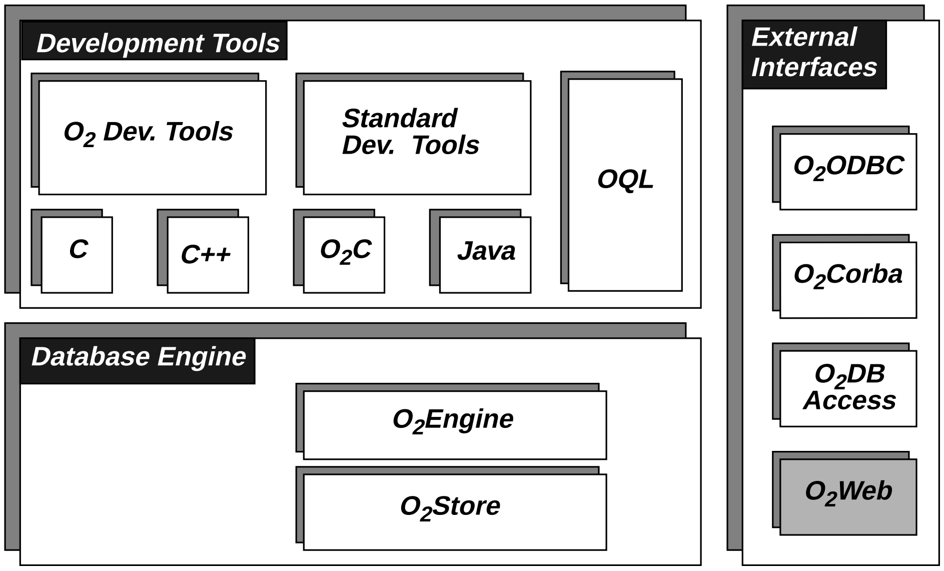
This chapter introduces O_2Web and is divided into the following sections:

- System overview
- The World Wide Web
- Features and advantages
- Manual overview

1 Introduction

1.1 System overview

O_2's system architecture is illustrated in the following figure.



O_2 System Architecture

The O_2 system consists of the database engine, development tools, and external interfaces.The database engine provides all the capabilities of a database and an object-oriented system. This engine is accessible by using programming languages, O_2 development tools, and other standard development tools. Numerous external interfaces are also provided.

The database engine consists of the following:

• O_2Store	The database management system (DBMS) provides low-level facilities, through O_2Store API, to access and manage databases: disk volumes, files, records, indexes, and transactions.
• O_2Engine	The object database engine provides direct control of schemas, classes, objects, and transactions through O_2Engine API. It provides full text indexing and search capabilities with O_2Search and spatial indexing and retrieval capabilities with O_2Spatial. It includes a Notification manager for informing other clients connected to the same O_2 server that an event has occurred, a Version manager for handling multiple object versions and a Replication API for synchronizing multiple copies of an O2 system.

Programming Languages:

You can create and manage O_2 objects by using the following programming languages:

• C	O_2 functions can be invoked by C programs.
• C++	ODMG-compliant C++ binding.
• Java	ODMG-compliant Java binding.
• O_2C	An object-oriented fourth-generation language specifically for developing object database applications.
• OQL	ODMG-standard, SQL-like object query language capable of handling complex O_2 objects and methods.

O_2 Development Tools:

• O_2Graph	Creates, modifies, and edits any type of object graph.
• O_2Look	Designs and develops graphical user interfaces, provides interactive manipulation of complex and multimedia objects.
• O_2Kit	Library of predefined classes and methods for fast development of user applications.
• O_2Tools	Complete graphical programming environment for designing and developing O_2 database applications.

Standard Development Tools:

You can use any standard programming language system, such as Visual C++ and Sun Sparcworks and any supported operating system.

External Interfaces:

• O_2Corba	Creates O_2-Orbix servers for accessing O_2 databases with CORBA.
• O_2DBAccess	Connects O_2 applications to relational databases on remote hosts; invokes SQL statements.
• O_2ODBC	Connects remote ODBC client applications to O_2 databases.
• O_2Web	Creates O_2 World Wide Web servers for accessing O_2 databases through the internet network.

1.2 The World Wide Web

The World Wide Web (WWW or Web) is a protocol for exchanging and distributing hypermedia information across the Internet network.

Since its creation at CERN in 1992, the Web has known a tremendous but unexpected success. There are now more than one million Web users, consulting more than 10000 Web sites, throughout the world.

Web servers provide interactive access to large amounts of complex, multimedia and distributed data including structured text, graphics, sound and video.

1.3 Features and advantages

As an object database system, O_2 has been specially designed to efficiently store and manage large amounts of complex and multimedia data.

O_2Web provides all the benefits of both database technology and object technology, in addition to a complete set of tools enabling rapid development and deployment of a Web server based on the O_2 system.

O_2Web provides Web clients with the ability to browse through hypermedia information stored in any O_2 database. The information presented to a Web client is a view of the objects in the database.

O_2 has been designed and tuned to deliver high performance when storing and manipulating large volumes of complex and multimedia data. O_2 allows you to address an unlimited amount of objects. The objects themselves can be very large binaries. O_2 also implements sophisticated management techniques such as buffering, indexing, clustering and query optimization.

Based on O_2, your Web server benefits from the following additional features:

- Information stored in O_2 can be physically reorganized without any application modification. This physical/logical independence is ensured by the use of an associative query language such as OQL.
- Data can be shared by several users connected either through the Web or through other means (a standard O_2 application).
- The O_2 system guarantees fast recovery from any kind of software or hardware failure.
- O_2 comes with a complete set of development tools allowing you to create classes and methods, to reuse and refine existing classes, to change them at any time.
- O_2 supports ready to use components to deal with audio/video/text data types and can easily interpret data from other sources.

1.4 Manual overview

This manual is divided into the following chapters:

- Chapter 1 - Introduction

  Introduces O_2Web and outlines some of its advantages.

- Chapter 2 - O_2Web Installation

  Describes how to install O_2Web.

- Chapter 3 - A World Wide Web Tour

  Gives an overview of the World Wide Web.

- Chapter 4 - O_2Web Overview

  Focuses on the differences between using the Web with file systems and using the Web with the O_2 system.

- Chapter 5 - Programming an O_2Web server

  Describes how to program an O_2Web server, with examples.

- Chapter 6 - O_2Web Reference

  Gives the complete details of all O_2Web commands.

2 O2Web Installation

This chapter describes how to install O_2Web and is divided into the following sections:

2 O2Web Installation

2.1 Requirements

In order to use O_2Web, you need a fully featured HTTP server.

O_2Web can be used with commercial servers (Netscape Communication Server, Netscape Commerce Server, Microsoft Information Server, etc.) and public domain servers (CERN httpd, NCSA httpd, etc.).

2.2 O2Web distribution

The O_2Web distribution contains the following:

- The **`o2web_gateway`** program. This is a CGI script that is called by the httpd server when an O_2 query is received.
- The **`o2web_server`** program. This program processes a query and returns the HTML result to the **`o2web_gateway`** gateway. The gateway then passes the result to the HTTP server.
- The **`o2open_dispatcher`** program. This program manages O_2Web activity. It knows every **`o2web_server`** running on your Local Area Network (LAN) and supplies each **`o2web_gateway`** with the address of the **`o2web_server`** best suited to answer a query.
- The O_2 schema named **`o2web`**. This schema contains the O2WebAssistant toolkit. It is delivered as a dump file located in **`$O2HOME/o2schemas/o2web.dump`**. This dump file must be loaded using **`o2dba_schema_load`** in an O_2 system in order to be used.
- A four step tutorial. This tutorial is installed in **`$O2HOME/samples/o2web/o2c`**. A README file in this directory explains how to use the examples.

2.3 Installation

This section describes the different steps required to install O_2Web.

Specifying a port number to access o2open_dispatcher

In order to be reachable by **o2web_server** and **o2web_gateway**, a port number must be associated to the **o2open_dispatcher** service. This is done by adding an entry to the operating system file that contains the available services on the network.

The service name associated with **o2open_dispatcher** is **o2opendispatcher**. The port number can be any number that is not used by another service. Depending on the operating system (UNIX or Windows NT), the new entry must be inserted into one of the following files:

- **/etc/services** - for UNIX.
- **$WINDIR\system32\drivers\etc\services** - for Windows NT.

The new entry in the file is as follows:

```
o2opendispatcher              7999/tcp
```

Retrieving the dispatcher host name

To access the **o2open_dispatcher**, **o2web_gateway** and **o2web_server** must retrieve the machine on which **o2open_dispatcher** is running. This can be achieved using the following three techniques:

1. Looking for a command-line argument. This technique is only used by **o2web_server** and overrides any other means of retrieving the dispatcher host name.
2. Looking for the **O2OPEN_DISPATCHER** environment variable. This technique is used by **o2web_server** if the dispatcher host has not been retrieved by the command line. It can also be used by **o2web_gateway** if it is able to retrieve this variable. Certain HTTP servers only pass CGI variables to a CGI script. Thus, this technique cannot be used with these types of HTTP server.
3. Looking for the content of the file **/etc/o2openaccess** (UNIX) or **$WINDIR\system32\drivers\etc\o2web** (Windows NT). This technique can be used by both **o2web_server** and **o2web_gateway** if the dispatcher host name has not been retrieved by any other means. This file must contain the dispatcher host name.

2.4 Launching O2Web

Running o2open_dispatcher

In order to start-up successfully, **`o2open_dispatcher`** needs the port number and the protocol used by the other programs to connect to it. This information is retrieved using the techniques described in the subsection Specifying a port number to access o2open_dispatcher.

This program has only one option (**`-v`**). This forces **`o2open_dispatcher`** to run in verbose mode.

It does not use any environment variables.

It must be running before **`o2web_server`** is launched.

Running o2server

Before running **`o2web_server`**, an **`o2server`** must be running. Consult the *O_2 System Administration Guide* for further details concerning **`o2server`**.

Running o2web_server

In order to start-up successfully, **`o2web_server`** needs the following information:

- The O_2 installation directory. This is given by the **`O2HOME`** environment variable (mandatory).
- The O_2 system to connect to. This is given by a directive in the **`.o2rc`** configuration file or by specifying a server hostname in the command line.
- The machine on which an **`o2server`** is running. This is given by a directive in the **`.o2rc`** configuration file or by specifying a server hostname in the command line.
- The machine on which an **`o2open_dispatcher`** is running. As explained in the subsection Retrieving the dispatcher host name, this information is retrieved by the **`o2web_server`** in a system-dependent file. It can be overridden by a directive in the **`.o2rc`** configuration file, or by specifying a dispatcher hostname in the command line.
- The port number and the protocol used to connect to **`o2open_dispatcher`**. As explained in the subsection Specifying a port number to access o2open_dispatcher, this information is retrieved by the standard operating system mechanisms.

`o2web_server` can run on any machine on your LAN.

2 O2Web Installation

An O_2 server running on the same system must be active before **`o2web_server`** can be run. The **`o2open_dispatcher`** program must also be running before **`o2web_server`** can be run.

Running an HTTP server

To test your O_2Web service, you need an HTTP server. Any HTTP server can be used (commercial or public domain). Your server must be configured to call the **`o2web_gateway`** CGI script when a URL leading to O_2 is received. This installation is specific to each HTTP server and cannot be explained here. Refer to your HTTP server documentation for details about mapping URLs to CGI scripts.

After configuration, the HTTP server will run an o2web_gateway process each time a URL leading to O_2 is received.

To run successfully, **`o2web_gateway`** needs the following information:

- The machine on which the **`o2open_dispatcher`** is running. As explained in the subsection Retrieving the dispatcher host name, this information is retrieved by **`o2web_gateway`** in a system-dependent file and can overridden by the **`O2OPEN_DISPATCHER`** environment variable when running HTTP servers that transfer their own environment to CGI scripts.
- The port number and protocol used to connect to **`o2open_dispatcher`**. This information is retrieved by standard operating system mechanisms, as explained in the subsection Specifying a port number to access o2open_dispatcher.

3 A World Wide Web Tour

This chapter gives an overview of the World Wide Web.

It contains the following sections:

3 A World Wide Web Tour

3.1 The World Wide Web

The World Wide Web (or Web or WWW) is a wide area client-server architecture for retrieving hypermedia information across the Internet. The Web has three main components:

- Universal naming scheme for documents. The Universal Resource Locator (URL) syntax specifies documents in terms of the protocol to be used to retrieve them, their Internet host and path name.
- Use of available protocols for retrieving documents over the network, including FTP, NNTP, WAIS, Gopher, and HTTP. The latter is designed specifically for use with the WWW, and combines efficiency with an ability to flexibly exchange information between clients and servers.
- A document format (HTML) supporting hypertext links based on URLs which can specify documents anywhere on the Internet. HTML is designed for rendering on a wide variety of different display types and platforms.

3.2 The HTML language

The Hypertext Markup Language (HTML) is the language used to write documents for the Web. Applications designed for the Web (usually called Web browsers) can read HTML documents and format them with text, graphics, tables and links to other HTML documents.

HTML is a markup language - in fact it is a specific implementation of the Standard Generalized Markup Language (SGML) - and is concerned with the structure of documents rather than their appearance. This feature gives HTML documents portability across different platforms or media. It is up to the browser reading an HTML document to map the structure into a physical format.

An HTML document is made of structure commands and plain text. The structure commands are called tags. A tag begins with a **<** and ends with a **>**. HTML tags can be either separator tags or container tags. A container tag is made of two parts: the beginning tag **`<X>`** and the ending tag **`</X>`**. The command specified is then applied to the text between the two tags. A separator tag is a "one shot" command; for instance the **`<hr>`** tag, which inserts an horizontal rule line, is a separator tag whereas the **`<b>`** tag, which makes the text contained between the **`<b>`** tag and the **`</b>`** tag bold, is a container tag.

3.3 Writing HTML documents

This section introduces the HTML language and the tags most commonly used for writing standard documents. It is not a complete HTML manual and is intended to help readers unfamiliar with the Web to understand the O_2Web basics.

An HTML document

Every HTML document has a common structure.

It declares itself as an HTML document with the **`<html>`** tag and ends with the **`</html>`** tag.

It comprises two main parts: the header and the body.

An HTML document has the following structure:

```
<html>
<head>
.....
</head>

<body>
.....
</body>
</html>
```

The header

The header content is not usually displayed by the browser in the document window. It contains information that can be either displayed in a separate window or when document information is requested. The most important command that must always be present in the header is the **`<title>`** command. This label is used, for example, when you store a reference to a document in your browser's "hotlist". Other information, such as the creator of the document, the creation date, etc., can also be written in the header.

```
<html>
<head>
<title> This is the title of my document </title>
</head>

<body>
.....
</body>
</html>
```

The body

The body of an HTML document contains both the structure and the content of the document.

Heading Levels

Heading tags are used to organize your document into a hierarchical structure. Different heading levels exist from level 1 to level 6. Each heading level puts text inside it with a particular font size, font attributes, etc.

Usually, the level 1 header (**<h1>**) is used for writing the title of your document, the level 2 header (**<h2>**) for the title of the document sections, the level 3 (**<h3>**) for the subsections, etc.

This kind of document structure is not forced by HTML, for which a header tag indicates only that the text between the header beginning tag and the header ending tag must have a particular style, but it is considered good practice to organize an HTML document in this way.

```
<html>
<head>
<title> This is the title of my document </title>
</head>

<body>
<h1> This is the title of my document </h1>

<h2> First Section </h2>

<h3> First SubSection </h3>

......
<h3> Second SubSection </h3>
......
<h2> Second Section </h2>
......
</body>
</html>
```

Paragraphs

The paragraph tag (**`<p>`**) is a separator tag. It cuts a piece of text into two different paragraphs. Most browsers when finding a paragraph tag, insert a blank line to separate the paragraphs. If you just want to break the current line, you can use the **`<br>`** tag.

```
<html>
<head>
<title> This is the title of my document </title>
</head>

<body>
<h1> This is the title of my document </h1>

<h2> First Section </h2>

This is the text of my first paragraph.
<p>
This is my second paragraph. It is cut here <br>
and continued on the next line.

......

</body>
</html>
```

Enumeration Lists

The list tags are very useful when the document must integrate enumeration of items. There are two kinds of lists: ordered lists and unordered lists.

When using ordered lists, each item appears preceded by its rank in the list; when using an unordered list, each item appears preceded by a marker (usually a bullet).

The tag corresponding to an ordered list is **`<ol>`** and the tag corresponding to an unordered list is **`<ul>`**. A separator tag (**`<li>`**) is used to indicate a new item in a list.

Enumeration list can be nested. In this case, visual outline effects are usually provided by most browsers.

```
<html>
<head>
<title> This is the title of my document </title>
</head>

<body>
<h1> This is the title of my document </h1>

<h2> First Section </h2>

......

<h3> First SubSection </h3>
<ol>
   <li> My first item
   <li> My second item
   <ul>
      <li> My first sub item
      <li> My second sub item
    </ul>
   <li> My third item
 </ol>

......

</body>
</html>
```

Definition Lists

A definition list is quite useful for expanding items in a list. The tag for defining a definition list is the **`<dl>`** tag. Each item in a definition list has:

a definition term (**`<dt>`**)

and definition data (**`<dd>`**) expanding or explaining the term

Definition lists are a very commonly used structure in HTML documents.

```
<html>
<head>
<title> This is the title of my document </title>
</head>

<body>
<h1> This is the title of my document </h1>

...

<h3> Second SubSection </h3>

<dl>

<dt> first item title
<dd> first item development or explanation

<dt> second item title
<dd> second item development. Note that the definition
data (such as any html tag content) can be as complex
as you want and contains any HTML construction. <p>
Here we just add a new paragraph.

 </dl>

......

</body>
</html>
```

Hypertext Links

Hypertext links are a key feature of HTML and are what makes the Web so exciting for many people. They are not only a way of pointing to another document stored on a disk but a way of pointing to another document located anywhere else on the internet network.

At this point, it is important to explain how a document located somewhere on the internet is retrieved by your favorite Web browser. The World Wide Web is based on a communication protocol, the HyperText Transfer Protocol (HTTP) and documents are referenced using something called a Uniform Resource Locator (URL). A simple URL might have the following format:

```
protocol://<machine internet address>[:port]/path/document
```

where:

- the **`protocol`** is usually HTTP but can also be another communication protocol such as FTP, NNTP, WAIS, Gopher, etc. In such cases, the URL is not used to retrieve an HTML document but for other purposes that do not enter into the scope of this manual,
- the **`machine internet address`** is an internet host on which an HTTP server is running,
- the optional **`port`** is the TCP port number on which the HTTP server is accessible. The standard HTTP port number is 80. This means that when accessing an HTTP server running on port 80, it is unnecessary to specify the port number in the URL,
- the **`path`** is something that will be mapped by the HTTP server (according to some configuration rules) into a physical path in its file system,
- the **`document`** is a reference to a file containing an HTML document.

To insert a link to another document in an HTML document you use the anchor tag (**`<a>`**). The anchor beginning tag contains an attribute specifying the URL of the linked document. The text between the anchor beginning tag and the anchor ending tag will be highlighted by the browser (usually underlined). The appropriate document will be retrieved when the user clicks on this text.

```
<html>
<head>
<title> This is the title of my document </title>
</head>

<body>
<h1> This is the title of my document </h1>

...

<h2> Second Section </h2>

The second section of the document is not here. It can
be retrieved <a href="http://xx.xx.xx/doc/sec2.html">
here. </a>

</body>
</html>
```

Inline Images

Another exciting feature of the Web is its multimedia orientation. Although documents integrate images, sound or video, only images and bitmaps can be put inline in HTML documents. The only image format that can be displayed by all browsers is the GIF format. It is therefore recommended that you use GIF images in your HTML documents. However, another format (JPEG) is becoming increasingly popular and can be put inline by many recent browsers.

An image is inserted in an HTML document by means of the **`<img>`** tag. This tag contains an attribute (**`src`**) specifying the URL of the image file.

The following optional attributes can be used with the **`<img>`** tag:

- the **`align`** attribute specifies the position of the image relative to the text. The possible values for this attribute are **`top`**, **`middle`** or **`bottom`**. Some browsers recognize other values but these are nonstandard.
- the **`alt`** attribute specifies a text label to be displayed instead of the image when a browser can not display images or when a browser is configured to only show images on demand.
- the **`ismap`** attribute specifies the image as a clickable image. This attribute is discussed in more detail in the section "clickable images".

Do not forget that inline images, although greatly improving the look of your documents, considerably slow down the retrieval of such documents. Keep this in mind when designing documents.

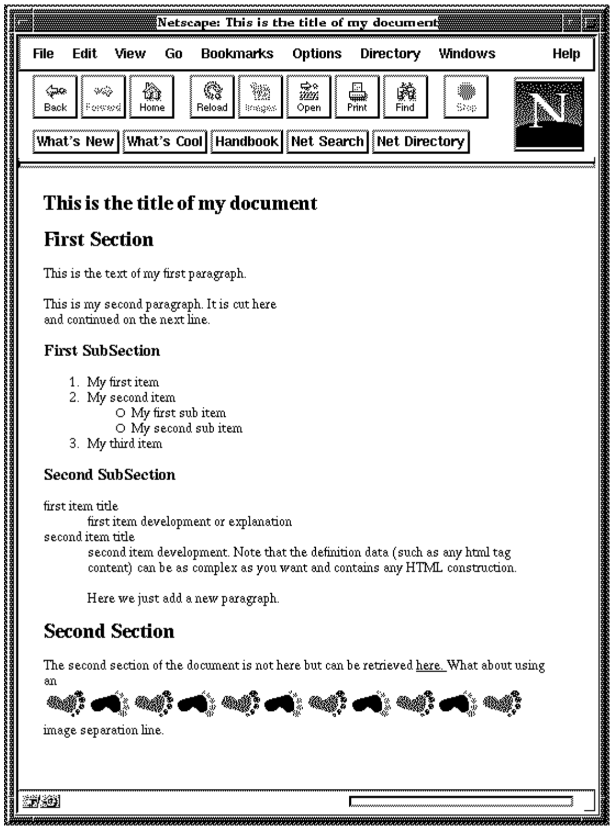
```
<html>
<head>
<title> This is the title of my document </title>
</head>

<body>
<h1> This is the title of my document </h1>

...

<h2> Second Section </h2>

The second section of the document is not here but can
be retrieved <a
href="http://www.o2tech.com/doc/sec2.html"> here.
</a>

What about using an <br>
<img src="http://xx.xx.xx/doc/step1.gif">
image separation line.

</body>
```

The complete document built so far has the following form:

```
<html>
<head>
<title> This is the title of my document </title>
</head>

<body>
<h1> This is the title of my document </h1>

<h2> First Section </h2>

This is the text of my first paragraph.
<p>
This is my second paragraph. It is cut here <br>
and continued on the next line.

<h3> First SubSection </h3>
<ol>
   <li> My first item
   <li> My second item
   <ul>
      <li> My first sub item
      <li> My second sub item
   </ul>
   <li> My third item
</ol>

<h3> Second SubSection </h3>

<dl>

<dt> first item title
<dd> first item development or explanation

<dt> second item title
<dd> second item development. Note that the definition
data (such as any html tag content) can be as complex
as you want and contains any HTML construction. <p>
Here we just add a new paragraph.

</dl>
```

```
<h2> Second Section </h2>

The second section of the document is not here but can
be retrieved <a
href="http://ww.o2tech.com/doc/sec2.html"> here. </a>
What about using an <br>
<img src="http://xx.xx.xx/doc/step1.gif">
image separation line.

</body>
</html>
```

This document will have the following appearance when viewed using a typical Web browser:



This is the title of my document

First Section

This is the text of my first paragraph.

This is my second paragraph. It is cut here
and continued on the next line.

First SubSection

1. My first item
2. My second item
   - My first sub item
   - My second sub item
3. My third item

Second SubSection

first item title
    first item development or explanation
second item title
    second item development. Note that the definition data (such as any html tag content) can be as complex as you want and contains any HTML construction.

    Here we just add a new paragraph.

Second Section

The second section of the document is not here but can be retrieved here. What about using an

image separation line.

Figure 3.1: A simple HTML page

Clickable images

A clickable image (or imagemap) is an inline image in which sensitive zones are defined. A URL is associated with each of these zones. The appropriate URL will be resolved when the user clicks on one of these zones. An imagemap is declared by the **`ismap`** attribute of the **`<img>`** tag.

Zones of an image are described in a file called a map file. Such a file contains, on each line, the specification of a region and its corresponding URL. The syntax of a map file is as follows:

- **`point URL x,y`** specifies a clickable point on the image. This is useful if a user clicks on an undefined area because the closest defined point is used.
- **`circle URL x,y x,y`** specifies a circle by the coordinates of both its center and any point on its circumference.
- **`rect URL x,y x,y`** specifies a rectangle by its upper left and lower right corners.
- **`poly URL x,y x,y ...`** specifies a polygon of up to 100 sides. Each (x,y) pair is a point where two sides of the polygon meet. The last point is always connected to the first one.
- **`default URL`** defines the default URL to be used if a user clicks on a point outside a defined region. When a point region is defined in the image, the default is never used.

Forms

HTML forms are a way of getting information from a user connected to your Web server. Forms can be used to insert buttons, input text fields, menus, etc. A form is something included inside the **`<form>`** and **`</form>`** tags. Two main attributes can be given to the **`<form>`** tag:

- **`action`** is the URL of a program to which the form content will be submitted. If this attribute is missing, the current document URL will be used. Forms usually use a special kind of URL that contains a path to a program instead of a path to a file. Such a program is called a CGI (Common Gateway Interface) script; this is a program that understands a very simple protocol allowing it to get information from the HTTP server. Writing a CGI script is out of the scope of this manual. To know more about writing CGI scripts, you should consult one of the many courses available on the internet. A good starting point is the Web Consortium server (**`http://www.w3.org`**).
- **`method`** is the HTTP method used to submit the fill-out form to a query server. The **`method`** attribute values can be **`get`** or **`post`**. When using the **`get`** method, the fill-out form content is appended to the URL; if the **`post`** method is used, the fill-out form contents are sent to the server in a data block. It is highly recommended to use post methods when writing forms as this technique does not have the limitations **`get`** methods can have when the fill-out form contents are large.

The form content is built using several tags:

- The input tag is used to specify a simple element inside a form. There is no corresponding ending tag. The possible attributes of the input tag are as follows:
  - **`type`** must be one of:

    `text`: a text entry field (default)

    `password`: a text entry field where typed characters are represented as asterisks.

    `checkbox`: a single toggle button; on or off.

    `radio`: a single radio button; on or off. Other radio buttons with the same name are grouped into "one of many" behavior.

    `submit`: a push-button that computes the fill-out form contents, formats it and resolves the URL in the **`action`** attribute of the form, appending the form contents to the URL (**`get`** method) or posting the form contents (**`post`** method).

    `reset`: a push-button that resets the different fields in the form to their default values.

    `hidden`: a hidden text field in the created HTML form. It is useful for contextual information.
  - **`name`** is the symbolic name for this input field. This is a mandatory attribute except for input tags of the type submit or reset. The name is used when building the formatted fill-out form contents transmitted to the server.
  - **`value`** can be used to specify the default value of a field (for a text or password input field). It can also be used to specify (for a checkbox or a radio button field) the value of a button when it is checked; the default value for a checkbox or radio button is "on". It can also be used to specify the label of a push-button.
  - **`checked`** specifies that a checkbox or a radio button is checked when displayed.
  - **`size`** is the size, expressed in characters, of text fields and password fields.
  - **`maxlength`** is the maximum number of characters accepted as input in text and password fields.
- The select tag is used to insert an options menu or scrollable lists into a form. It is used as follows:

```
<select name="my-menu">
     <option> first item
     <option> second item
     <option> third item
</select>
```

The possible attributes of the **`select`** tag are:

- **`name`** is the symbolic name for this element. This is a mandatory attribute of the tag. The name is used when building the formatted fill-out form contents transmitted to the server.
- **`size`** is a specified integer. When its value is 1 (or if the attribute is missing) then the form element will be represented as an options menu. If **`size`** is greater than 1, the form element will be represented as a scrollable list. The value of **`size`** determines how many items are visible in the window.
- **`multiple`** specifies that the user can make multiple selections ("n of many" behavior). This attribute forces the form element to be a scrollable list regardless of the **`size`** attribute value.

• The **`textarea`** tag can be used to insert a multi-line text area in a form.

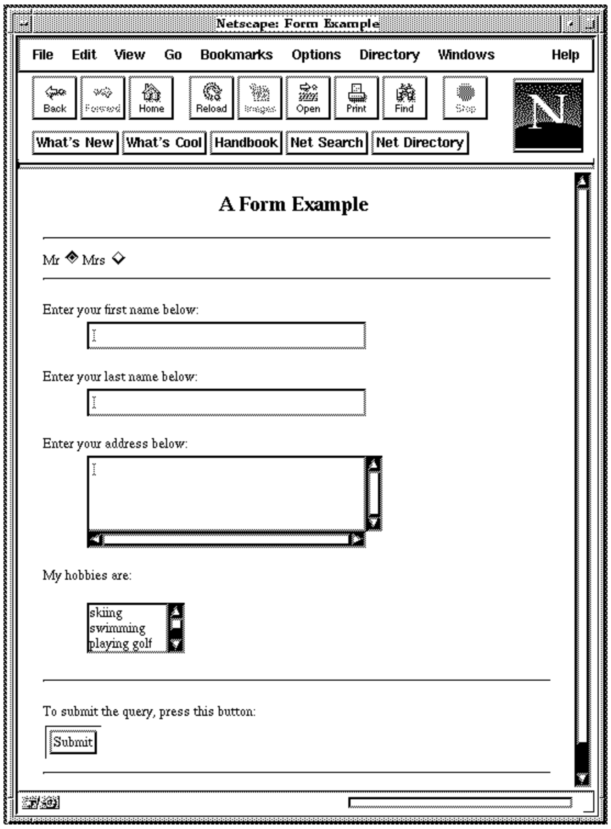
• The attributes of a **`textarea`** are as follows:

- **`name`** is the symbolic name of the field
- **`rows`** is the number of rows in the text area
- **`cols`** is the number of columns in the text area.

The following example shows a complete form:

```
<html>
<head>
<title> Form Example </title>
</head>
<body>
<h1><center> A Form Example </center></h1>
<form method="POST"
action="http://www.site.domain/cgi/getperson">

<hr>
Mr   <input type ="radio" value="Mr" name="KIND" checked
>
Mrs <input type ="radio" value="Mrs" name="KIND">
<hr> <p>

<dl>
<dt>Enter your first name below:
<dd><input values="" size="40" name="FN">
<p>
<dt>Enter your last name below:
<dd><input values="" size="40" name="LN">
<p>
<dt>Enter your address below:
<dd><textarea rows=5 cols=40 name="ADDR"></textarea>
</dl>

<dl>
<dt> My hobbies are: <p>
<dd> <select name="HOBBIES" multiple size=3>
<option> skiing
<option> swimming
<option> playing golf
<option> playing tennis
<option> chess
</select> <p>
</dl>
<p> <hr> <p>
<p>
To submit the query, press this button:<br>
<input type="submit" value="Submit">
<hr>
</form>
</body>
</html>
```

This document will have the following appearance when viewed by a typical Web browser:


Netscape: Form Example
File Edit View Go Bookmarks Options Directory Windows Help
Back Forward Home Reload Images Open Print Find Stop
What's New What's Cool Handbook Net Search Net Directory
Submit

A Form Example

Mr Mrs

Enter your first name below:

Enter your last name below:

Enter your address below:

My hobbies are:

skiing
swimming
playing golf

To submit the query, press this button:

Figure 3.2: An HTML form example

In the above example, when the user clicks on the submit button, the URL specified in the action attribute will be resolved and the specified CGI script will be executed. As the method is a post method, a data block containing the result of the form will be sent to the server. It is up to the CGI script to read this data block. In this example, the following data will be sent:

```
KIND=Mr&FN=John&LN=Smith&ADDR=this%20is%20an%20address
&HOBBIES=skiing&HOBBIES=chess
```

In the data block above, some specificities of the format generated by a fill-out form are as follows:

- Each value retrieved in the form is built as **`name=value`** where **`name`** is the attribute name of the field and **`value`** is either the value entered by the user or the default value. All the input field results are put together, separated by the **&** character. For a multiple list input field, each selected value is repeated **`name=value1&name=value2`**,
- Some characters are replaced by codes. This is discussed in the following section.

3.4 Special characters in HTML text

Some characters have special meaning within HTML and therefore cannot be used "as is" in text. These characters are, for example, the angle bracket or the quote characters.

As the Web is multiplatform, only a reduced set of characters can be typed in a text. Namely, only lower ASCII characters (all the characters of an English keyboard) can be used. Upper ASCII characters cannot be used directly and must be "escaped".

An escape sequence can be either character references or entity references.

A character reference has the format &#nnn, where nnn is a number that references the character.

An entity reference has the format &nnn, where nnn is a text string that references the character.

The following table summarizes the special characters in HTML text.

TABLE 3.1 Special characters in HTML text

Character	Reference	Entity
"	&#34	&quot
&	&#38	&amp
<	&#60	&lt
>	&#62	&gt
ª	&#170	&ordf
«	&#171	&laquo
¬	&#172	&not
-	&#173	&shy
®	&#174	&reg
¯	&#175	&macr
°	&#176	&deg
	&#177	&plusmn
	&#178	&sup2
	&#179	&sup3
´	&#180	&acute
	&#181	&micro
¶	&#182	&para
·	&#183	&middot

¸	&#184	&cedil
	&#185	&sup1
º	&#186	&ordm
»	&#187	&raquo
	&#188	&frac14
	&#189	&frac12
	&#190	&frac34
¿	&#191	&iquest
À	&#192	&Agrave
Á	&#193	&Aaute
Â	&#194	&Acirc
Ã	&#195	&Atilde
Ä	&#196	&Auml
Å	&#197	&Aring
Æ	&#198	&AElig
Ç	&#199	&Ccedil
È	&#200	&Egrave
É	&#201	&Eacute
Ê	&#202	&Ecirc
Ë	&#203	&Euml
Ì	&#204	&Igrave
Í	&#205	&Iacute
Î	&#206	&Icirc
Ï	&#207	&Iuml
	&#208	&ETH
Ñ	&#209	&Ntilde
ò	&#210	&Ograve
ó	&#211	&Oacute
ô	&#212	&Ocirc
õ	&#213	&Otilde
ö	&#214	&Ouml
	&#215	
Ø	&#216	&Oslash
Ù	&#217	&Ugrave
Ú	&#218	&Uacute

Û	&#219	&Ucirc
Ü	&#220	&Uuml
	&#221	&Yacute
	&#222	&THORN
ß	&#223	&szlig
à	&#224	&agrave
á	&#225	&aacute
â	&#226	&acirc
ã	&#227	&atilde
ä	&#228	&auml
å	&#229	&aring
æ	&#230	&aelig
ç	&#231	&ccedil
è	&#232	&egrave
é	&#233	&eacute
ê	&#234	&ecirc
ë	&#235	&euml
ì	&#236	&igrave
í	&#237	&iacute
î	&#238	&icirc
ï	&#239	&iuml
	&#240	&eth
ñ	&#241	&ntilde
ò	&#242	&ograve
ó	&#243	&oacute
ô	&#244	&ocirc
õ	&#245	&otilde
ö	&#246	&ouml
	&#247	
ø	&#248	&oslash
ù	&#249	&ugrave
ú	&#250	&uacute
û	&#251	&ucirc
ü	&#252	&uuml
	&#253	&yacute

	&#254	&thorn
ˇ	&#255	&yuml

3.5 Special characters in URLs and form submissions

Section 2.4 was concerned only with characters witin HTML text. Another kind of encoding must be used within a URL or to decode a string resulting from an HTML form submission.

In a URL, a character can be encoded by the percent character (%) followed by the hexadecimal value of the character. A character must be encoded if it has no corresponding graphic character in the US-ASCII character set, if the use of the corresponding character is unsafe, or if the corresponding character is reserved for a specific purpose.

No corresponding graphic in US-ASCII

The characters between 7F and FF hexadecimal and control characters (between 00 and 1F) must be encoded.

Unsafe characters

The unsafe characters are:

```
space < > « # % { } | \ ^ ~ [ ]  `
```

Reserved characters

The reserved characters are:

```
; / ? : @  = &
```

3.6 HTML tags summary

Previous sections introduced the main tags in the HTML 2.0 specification. Many others tags exist and many new ones may appear as HTML evolves. For example, many recent browsers such as Netscape Navigator© introduce tags that are part of the HTML 3.0 specification. These tags relate to tables, text centering and font size.

The table below summarizes the main tags in the HTML 2.0 specification.

TABLE 3.2 The main tags in the HTML 2.0 specification

Tag name	Description
HTML	Defines the file as an HTML document
HEAD	Defines the heading for the document
TITLE	Defines the title of the document
BODY	Defines the document body
H1...H6	Defines the six levels of headings
P	Defines a paragraph
OL	Defines an ordered list
UL	Defines an unordered list
DIR	Defines an unordered list with several items per line
MENU	Defines an unordered list, typically with one line per item
LI	Defines an individual item in a list
DL	Defines a definition list
DT	Defines an individual definition term of a definition list
DD	Defines individual definition data of a definition list
EM	Emphasizes characters (usually italic)

STRONG	Strongly emphasizes characters (usually bold)
CODE	Displays text in a fixed-width font (usually courier)
B	Displays text in bold
I	Displays text in italics
TT	Displays text in typewriter-like font
IMG	Inserts an image in the document
HR	Inserts a horizontal rule line
BR	Breaks the current line
A	Defines an anchor to another document
PRE	Inserts preformatted text
ADDRESS	Inserts text in address-like formatting (italic, smaller font)
BLOCKQUOTE	Defines text quoted from another source

3 A World Wide Web Tour

4 O2Web Overview

This chapter focuses on the differences between using the Web with file systems (introduced in the previous section) and using the Web with the O_2 system.

It contains the following sections:

4 O2Web Overview

4.1 Principles

A user connecting to a Web server built on top of O_2Web directly accesses objects in an O_2 database rather than files.

To specify an object to be retrieved in the database, a Web client specifies an OQL query rather than a directory path to a file.

OQL is the standard object query language, defined by the ODMG (Object Database Management Group) to query object databases. As is an object extension of SQL, OQL allows complex object databases to be queried and object methods to be invoked.

For example, the query:

```
select distinct employee.salary
from employee in All_employees
where employee.name like "Mac*"
```

returns the salary (or salaries) of all the employee(s) whose name begins with **`Mac`**.

For further information about OQL, please refer to either the OQL User Manual from O_2 or the ODMG-93 standard[1].

To make OQL queries enter in the framework of URLs, O_2Web uses the same kind of URL as the one used by HTML fill-out forms. These URLs have the following form:

```
http://host[:port]/path/script/[extrapath][?search]
```

This is the more general form of URLs where:

- **`host`** is the internet host on which an HTTP server is running,
- **`port`** is the TCP port number on which the server is accessible,
- **`path`** is a logical path translated by the HTTP server into a physical path on the file system,
- **`script`** is the name of a program compliant with the CGI protocol,

1. The Object Database Standard: ODMG - 93. Atwood, Duhl, Ferran, Loomis and Wade. Edited by R.G.G. Cattell. Copyright 1994 Morgan Kaufmann Publishers.

- **`extrapath`** is a set of strings (separated by the / character) that can be retrieved by the script,
- **`search`** is a string that can be retrieved by the script.

When a Web server is built on top of O_2Web, a URL that accesses this server must comply with the above URL form. The **`script`** component is a program (**`o2web_gateway`**) provided with O_2Web, the **`extrapath`** component contains the name of an O_2 system and the name of an O_2 database. The **`search`** component contains an OQL query. The result of this query is an HTML view of an object in the database.

A key feature of O_2Web is that it can be used at different levels. Starting from a completely automatic mode where HTML is generated for any object of the database, the programmer can progressively improve the Web service by overloading the generic mode for some classes of a schema.

4.2 O2Web Architecture

O_2Web comprises three main elements which are used with a standard full featured WWW server (Netscape server, CERN HTTPD, NCSA HTTPD, etc.):

- An O_2Web gateway.
- An O_2Web server.
- An O_2open dispatcher.

An O_2Web server may be accessed from any standard Web client (Mosaic, Netscape, ...).

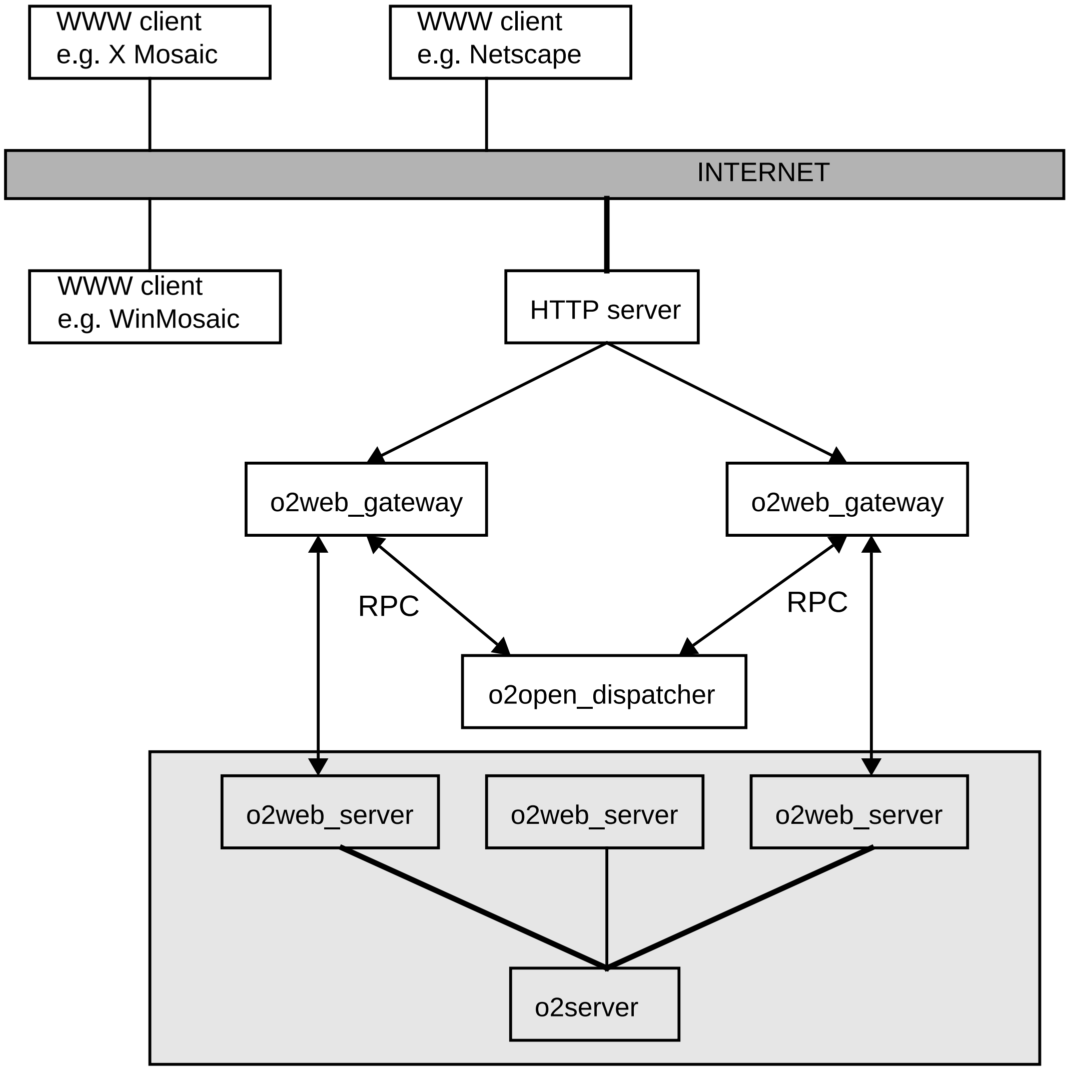


Figure 4.1: O_2Web Architecture

O2Web Architecture

The O_2Web system works in the following way:

1 A Web client sends a URL in HTTP format. This URL contains an OQL query.

2 The HTTP server passes the query to the O_2Web gateway.

3 The O_2Web gateway connects to an O_2Web dispatcher running on your local area network.

4 The O_2open dispatcher tells the O_2Web gateway which O_2Web server to connect to.

5 The O_2Web gateway connects to the appropriate O_2Web server.

6 The O_2Web server runs the query specified in the URL and transforms the result of the query into HTML (or other formats such as GIF and BITMAP when required).

7 The resulting data is sent back to the Web client.

4 O2Web Overview

5 Programming an O2Web Server

This chapter describes how to program an O_2Web server.

It is divided into the following sections:

5 Programming an O2Web Server

5.1 Introduction

O_2Web can be used at different levels.

The first (and simplest) level is to let O_2Web generate HTML for the result of the query contained in the URL. This is the generic mode of O_2Web.

The second level allows programmers to globally change parts of the HTML generation. The data retains the same generic look, but HTML text may be inserted at the top or at the bottom of pages. This level also allows you to react to some events such as a connection or a disconnection to the server (in order to maintain a log in the database), or when an error occurs (in order to personalize the error message the client will receive).

The last level allows total control of HTML generation for each class of the working schema. This means that a programmer can specify HTML text for the objects of each class. As this text is built by a method, it can be made to depend on the values of objects. The programmer can also define, for each class, the HTML text that will be inserted at the top and at the bottom of each page when an object of this class is the entry point of an HTML report.

5.2 Generic mode

Simple Browsing

The first step when one wants to provide a Web interface to an O_2 database is very simple. Just install O_2Web on your system (refer to Chapter 1 for installation details). You are ready to browse your database with a Web browser.

To begin browsing, you must provide your browser with a URL (or click on an already existing link found somewhere on the internet). The kind of URLs given to start a browsing session usually point to a persistent root of the database. Such a URL could be:

```
http://xx.xx.xx/cgi/o2web_gateway/sysname/basename?rootname
```

This URL will return an HTML view of the specified root of persistence. You can now click on the links on this page to continue browsing.

How does it work

Let us now explain how the generated HTML is built.

System-supplied methods can generate HTML for any object in the database, however complex it is. The generated HTML is based on the structure type of the object being processed. The exact behavior of the system-supplied methods is as follows:

- atomic values: integers, chars, booleans, reals, strings, bytes. The value is printed except for bytes which are not displayed.
- tuple values: A **`<ul>`** HTML list is used, each attribute of the O_2 tuple being an item in the list. Each item is composed of an attribute name and the result of the recursive call to an HTML production method on an attribute value.
- collection values: lists, sets. A **`<ul>`** HTML list is used, each element in the O_2 collection being printed as an item of the list. Each item is composed of the result of the recursive call to an HTML production method on a collection element.
- objects: An object is printed by the recursive call to an HTML production method on the encapsulated value.
- sub-objects: An HTML anchor is generated with a reference to a URL that contains an OQL query returning this sub-object.

You obtain the text of the anchor by calling the **`html_title`** method on the subject. If this method does not exist, the name of the class is used.

The subclasses of O_2Kit classes have particular printing methods:

- The class Date: Although the encapsulated value is a tuple, objects are printed as dates.
- The class Text: Although the encapsulated value is a list, objects are printed as plain text.
- The class Bitmap: Objects are printed as inline Bitmaps.
- The class Image: Objects are printed as inline GIF images.

A Guided Example

Let us take an example. We define in the database the schema of a very simple phone book. The aim of this example is to demonstrate how bases of this schema can be made accessible from the Web. The schema and the method code of this simple application can be found in the O_2 installation in the **`$O2HOME/samples/o2web/o2c/step1/`** directory. The implementation of some methods, unnecessary for the purposes of this example, will not be detailed in this manual. For further information concerning these methods, please refer to the samples directory of O_2.

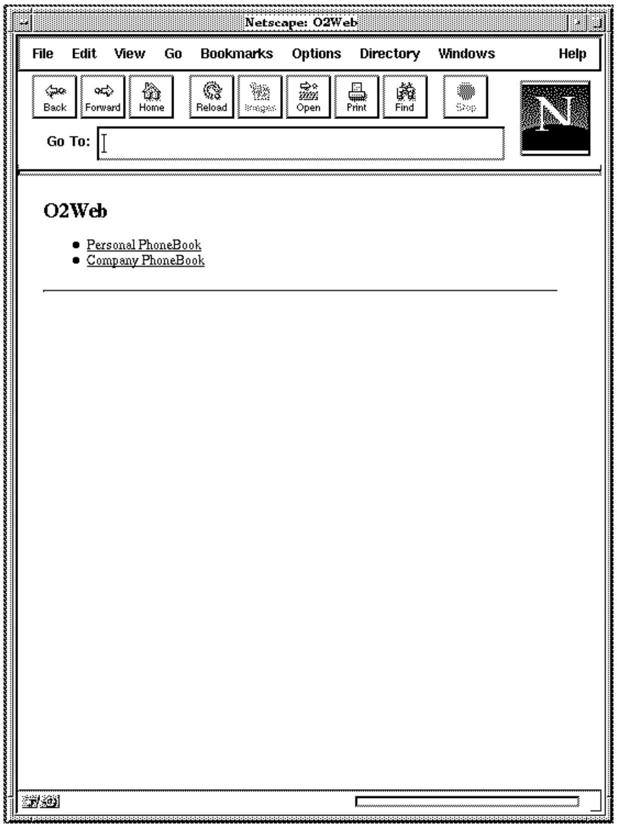
The schema of the application is as follows:

First, we import the class Image from the **`o2kit`** schema. We create a sub-class of the imported class in order to be able to add methods to this class.

```
import schema o2kit class Image;
rename class Image as ImportedImage;
class Image inherit ImportedImage end;
```

Then we create the **`Directories`** class. This is the container of all the directories we will create in the database. A persistent root is defined in this class. This name will be the entry point in the database for retrieving all the information.

```
class Directories public type
set(Directory)
end;
method public add_directory(name:string):Directory in class
Directories;

name DIRECTORIES : Directories;
```


Netscape: O2Web
File Edit View Go Bookmarks Options Directory Windows Help
Back Forward Home Reload Images Open Print Find Stop
Go To:

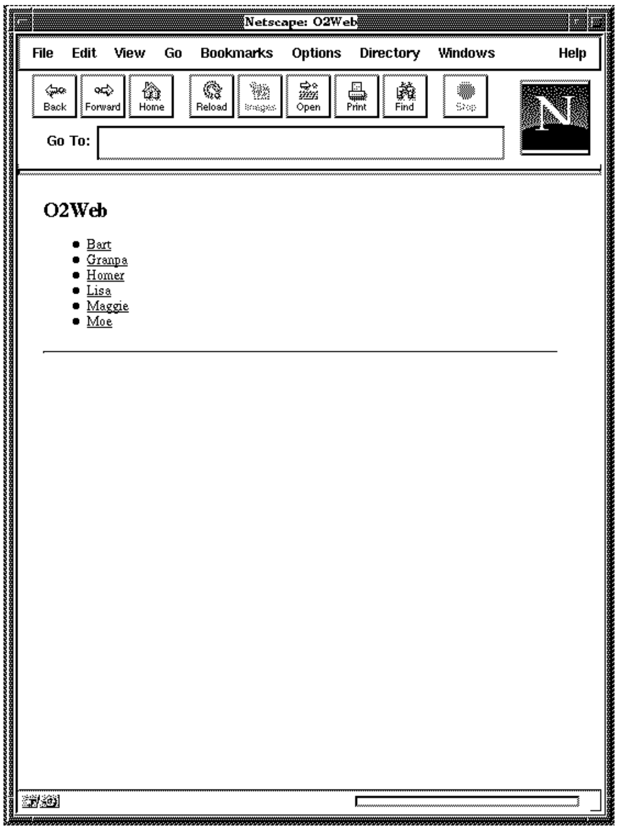
O2Web

- Personal PhoneBook
- Company PhoneBook

Figure 5.1: The Directories class

We now define the **`Directory`** class. This has a name and an attribute called **`entries`**. This attribute refers to the **`Entries`** class whose type is an enumeration of **`Entry`**.

```
class Directory public type
tuple(name : string,
      entries : Entries)
end;
method init(name:string) in class Directory;
method public html_title:string in class Directory;
method public new_entry(name:string, photo: Image,
                        address:string,phone:string,
                        e_mail:string) in class Directory;

class Entries public type
list(Entry)
end;
method public insert(d:Directory,entry:Entry)
in class Entries;
```



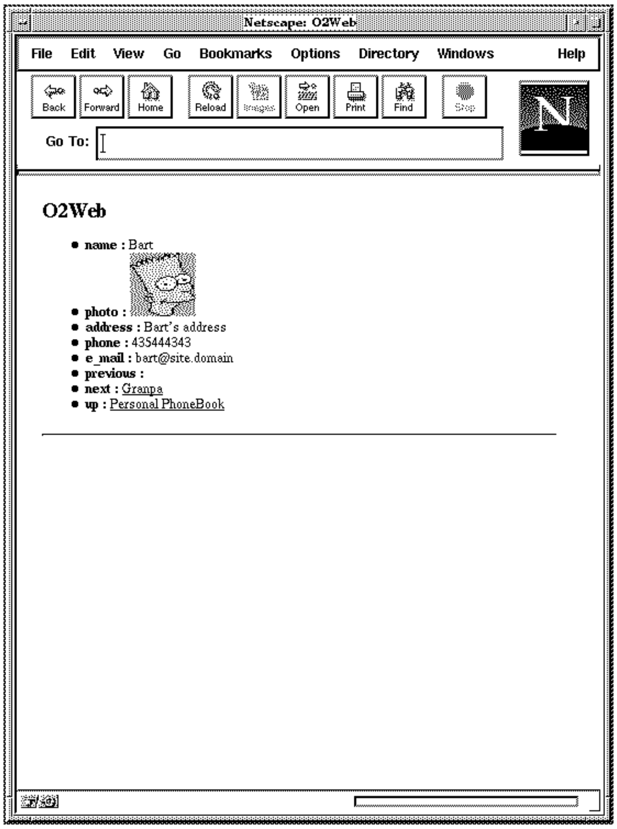
O2Web

- Bart
- Granpa
- Homer
- Lisa
- Maggie
- Moe

Figure 5.1: The Entries class

The **`Entry`** class contains an entry in a directory. An entry has a name, a photo, an address, a phone number, an e-mail address, and three attributes (**`previous`**, **`next`** and **`up`**) which respectively refer to the previous entry in the directory, the next entry in the directory and the directory itself. These attributes will be used to browse through a directory.

```
class Entry public type
tuple(name : string,
      photo : Image,
      address : string,
      phone : string,
      e_mail : string,
      previous : Entry,
      next : Entry,
      up : Directory)
end;
method public title:string in class Entry;
```



O2Web

- **name :** Bart
- **photo :**
- **address :** Bart's address
- **phone :** 435444343
- **e_mail :** bart@site.domain
- **previous :**
- **next :** Granpa
- **up :** Personal PhoneBook

Figure 5.1: The Entry class

Details of the method bodies for this schema are not given as they do not contain O_2Web-relative code. You can consult these methods in the **`samples/o2web/o2c/step1`** directory of the O_2 distribution.

After loading this schema, the associated methods and some data in O_2, you are ready to browse through the phone book. You can see that the result is a good starting point to help you evaluate your final service.

5.3 Global Personalizations

The generic mode of O_2Web can be considered as a great prototyping tool in the process of building a large scale Web service. However, there is a need for more sophisticated mechanisms in order to build a personalized Web service. This section explains how to globally personalize parts of a generic service.

The global personalizations are handled by means of a named object whose name is **`TheO2WebInteractor`**. This object must belong to the **`O2WebInteractor`** class or one of its subclasses. Global personalizations are achieved by defining methods in the **`O2WebInteractor`** class. O_2Web, in the process of generating HTML, will call these methods on the **`TheO2WebInteractor`** root if they are defined.

Adding headers

To add a constant header, the programmer must define a method called **`header`** in **`TheO2WebInteractor`** root. O_2Web checks for this method definition before calling the generic HTML generation. If it is defined the result is inserted before the standard HTML.

Adding footers

To add a constant footer, the programmer must define a method called **`footer`** in **`TheO2WebInteractor`** the root. O_2Web checks for that method definition before calling the generic HTML generation. If it is defined the result is inserted before the standard HTML.

Changing the default prolog and epilog

The default prolog and epilog can be modified. This is usually unnecessary to do so because the default prolog contains the MIME type for the returned document and should only be changed globally for a class returning a specific MIME type (a class that handles postscript documents for instance). However, this method can be overloaded just in case the HTML prolog format changes in future versions of HTML.

React to a connection

When an O_2Web server is contacted to answer a query, the existence of the **`connect`** method at the **`TheO2WebInteractor`** root is verified. If it exists, **`connect`** is called to allow the programmer to perform certain actions. The method **`connect`** must be defined as follows:

```
method public connect(query:string,userdata:string):boolean
```

where

- **`query`** is the query given to the O_2Web server and for which a HTML report has to be built.

- **`userdata`** is a string a programmer can store in a URL to be returned when certain methods are called. It allows the programmer to store context dependent information when a URL is created, and to enable specific processing according to this information when the URL is resolved (for more information about **`userdata`**, see the **`make_url`** method in the O_2WebAssistant Toolkit).

If this method returns **`true`**, the connection is accepted. A return value of **`false`** will reject the connection.

React to a disconnection

After generating HTML text in response to a user query, O_2Web checks for the existence of the **`disconnect`** method on the **`TheO2WebInteractor`** root. If it exists, **`disconnect`** is called allowing the programmer to perform certain logging actions. The method **`disconnect`** must be defined as follows:

```
method public disconnect(size:integer,kind:string)
```

where

- **`size`** is the number of bytes returned for that connection and

- **`kind`** is the MIME type of the returned text.

Make your own error messages

When an error occurs, an error message is returned to the Web browser. You may want to change these messages for your server or enhance them by adding images to the error message text.

This can be done by defining the **`error`** method on the **`TheO2WebInteractor`** root. If defined, this method must return the complete HTML text, including the content type, to the client. The signature of this method is as follows:

```
method public error(kind:integer):bits
```

The **`kind`** parameter is one of the possible error codes returned by O_2Web. These codes are defined in the file **`o2web.h`**.

Enforced Encapsulation

You may need to restrict the objects or portions of objects that are revealed to clients, and you can do so by defining custom methods for an object or for an entire class.

When a client submits a query, O_2Web evaluates it and builds an HTML stream by using O_2's default HTML generation tools, or by using custom methods that you write. However, because methods can be applied to objects only, and not to values, you need to be able to control the type of results generated in response to user queries.

To support only queries whose results are objects, define the **`value_report`** method on the **`O2WebInteractor`** class, as follows:

```
method public value_report: bits
```

This method returns an error message when an query requesting a value is received by the O_2Web server.

A Guided Example

We will now modify our previous example in order to add text to the top and bottom of every page. We will also personalize the error messages. The extra-code for this example can be found in the **`samples/o2web/o2c/step2`** directory of the O_2 distribution.

We import the class **`O2WebInteractor`** and create a sub-class in order to be able to add methods to this class. Three methods are defined in the **`O2WebInteractor`** class. These methods are automatically called by O_2Web at the beginning of HTML generation (**`header`**), at each ending of HTML generation (**`footer`**) and each time an error will occur (**`error`**).

We also define a persistent root called **`TheO2WebInteractor`**; It is mandatory to create this name for the methods of the class to be called.

```
import schema o2web class O2WebInteractor, O2WebAssistant;
rename class O2WebInteractor as ImportedO2WebInteractor;

class O2WebInteractor inherit ImportedO2WebInteractor end;
method public header:bits in class O2WebInteractor;
method public footer:bits in class O2WebInteractor;
method public error(k:integer):bits in class O2WebInteractor;

name TheO2WebInteractor: O2WebInteractor;
```

Let us have a look at these three methods.

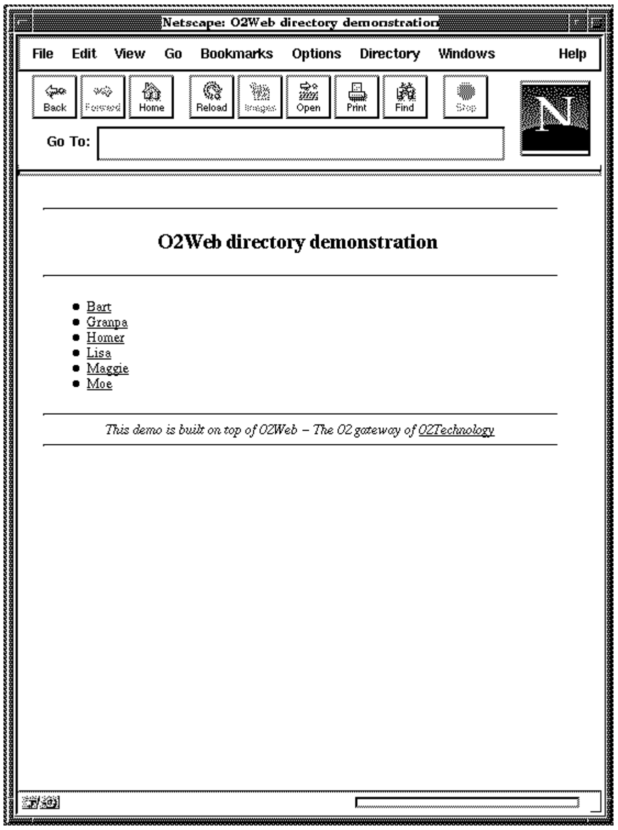
The **header** method returns the HTML text to be inserted before each new HTML generation.

This is a simple header containing formatted HTML text. Notice the tag **CENTER** in the code below. This tag is not recognized by all browsers (only Netscape recognizes it), so it must be used carefully.

```
method body header:bits in class O2WebInteractor
{
      o2 bits header;
      header += "<html><header>\n";
      header +=
            "<title> O2Web directory demonstration</title>\n";
      header += "</header><body>\n";
      header += "<HR>\n";
      header += "<CENTER>\n";
      header += "<H2> O2Web directory demonstration </H2>\n";
      header += "</CENTER>\n";
      header += "<HR>\n";
      return header;
};
```

In the **footer** method, we build a text containing an address and insert an anchor pointing to an O_2Technology Web server.

```
method body footer:bits in class O2WebInteractor
{
  o2 bits footer;
  footer += "<HR>\n";
  footer += "<ADDRESS>\n";
  footer += "<CENTER>\n";
  footer += "This demo is built on top of O2Web - The O2 gateway
of <a href=http://www.o2tech.fr> O2Technology</a>";
  footer += "</CENTER>\n";
  footer += "<HR>\n";
  footer += "</body></html>";
  return footer;
};
```


Netscape: O2Web directory demonstration
File Edit View Go Bookmarks Options Directory Windows Help
Back Forward Home Reload Images Open Print Find Stop
Go To:

O2Web directory demonstration

- Bart
- Granpa
- Homer
- Lisa
- Maggie
- Moe

This demo is built on top of O2Web – The O2 gateway of O2Technology

Figure 5.1: The global header and footer

The **`error`** method is defined below. Notice the CGI header, containing the MIME type of the returned document, in the HTML text.

```
method body error(k:integer):bits in class O2WebInteractor
{
      o2 bits error;
      o2 O2WebAssistant toolbox = new O2WebAssistant;

      error += toolbox->get_http_prolog("text/html");
      error += "<HR>\n";
      error += "<CENTER>\n";
      error += "<H2>Hypertext Documentation Error</H2>\n";
      error += "</CENTER>\n";
      error += "<HR>\n";
      error += "<H3>\n";
      error += "An error has occurred, please try again.<p>";
      error += "If the error persists, contact Mr Patch
(patch@rescue.com)";
      error += "</H3>";
      error += "<HR>\n";
      return error;
};
```

5.4 Local Personalizations

Local personalizations allow a programmer to control all aspects of the HTML returned by a query. Local personalization involves the definition of a method in a class that will be in charge of producing a part of the HTML.

With the exception of the events handling methods (**`error`**, **`connect`** and **`disconnect`**), all the other methods that can be defined in the **`O2WebInteractor`** class can be overloaded locally in any class of a schema.

Adding a header to the top of a page

The programmer can decide that all queries received by an O_2Web server returning an object of a particular class must have a specific header that depends on the data contained in the object or that is well adapted to the meaning of that class.

This is done by defining a method called **`html_header`** in a class of the schema. If there was a **`header`** method defined in the **`O2WebInteractor`** class, it is not used for classes in which a specific **`html_header`** method is defined.

Adding a footer to the bottom of a page

The programmer can decide that all queries received by an O_2Web server returning an object of a particular class must have a specific footer that depends on the data contained in the object or that is well adapted to the meaning of that class.

This is done by defining a method called **`html_footer`** in a class of the schema. If there was a **`footer`** method defined in the **`O2WebInteractor`** class, it is not used for classes in which a specific **`html_footer`** method is defined.

Changing the prolog and epilog

Suppose that you have created a class that encapsulates data of a specific format (postscript document, JPEG image, MPEG movie, etc.).

You must inform the HTTP server that the data you send to it has a specific format. The method **`html_prolog`** can be defined in this class and returns the MIME type corresponding to the format handled by the class.

For instance, a class managing MPEG movies will define a method **`html_prolog`** returning the string:

```
Content-type: movie/mpeg\n\n
```

Having a specific method to handle CGI headers allows a programmer to define a class managing any binary format and to create a subclass for each format. Thus, the only method defined in subclasses is the **`html_prolog`** method.

Building the body of a report

The body of a report can only be redefined locally in a class. The programmer can define a method called **`html_report`** in a class and the result of this method will be used instead of calling the generic HTML production.

Optimizing the query generation

When the programmer lets O_2Web generate queries for the sub-objects of an object, these queries are built starting from the query leading to the current object and adding to it a selection predicate or the selection of an attribute. In certain classes, the programmer may decide to generate the queries leading to the instances of the classes. This is achieved by defining a method called **`get_query`**.

A guided example

In this section, we will improve our example in two phases: in the first phase, we optimize the generation of queries of the generic mode. In the second phase, we completely customize the HTML generation.

Optimizing the query generation

In the two previous examples (generic mode and global personalization), the query generated and inserted in the anchors grew in size each time a user browsed. A query inserted in a link was built automatically by O_2Web, starting from the current query and adding a selection predicate or an attribute name to it. This can lead O_2Web to generate very large queries.

This problem can be avoided if the programmer knows how to directly reach objects of a class from the persistent roots of the schema. In this case, we define **`get_query`** methods in the classes **`Directories`**, **`Directory`** and **`Entry`**. When **`get_query`** is applied to an object, the text of an OQL query returning the object is returned. O_2Web uses this query instead of generating a query.

The code for these methods can be found in the **`samples/o2web/o2c/step3`** directory of the O_2 distribution.

```
method body get_query:string in class Directories
{
      return "DIRECTORIES";
};

method body get_query:string in class Directory
{
      return (o2 string)
            "element(select d from d in DIRECTORIES
            where d->html_title = "+ "\"" + self->html_title + "\")";
};

method body get_query:string in class Entry
{
      return (o2 string)
            "first(select e from e in " + self->up->get_query
            + "->entries where e->html_title = "
            + "\"" + self->html_title + "\")";
};
```

After defining these three methods, you can now browse a document and see that the queries contained in the HTML anchors no longer grow when you click on the links.

A complete customization

The code of the complete customization can be found in the **`samples/o2web/o2c/step4`** directory of the O_2 distribution.

We import, from the **`o2web`** schema, two classes in the O_2WebAssistant Toolkit. These classes are used to inline images and retrieve information from an HTML form.

```
import schema o2web class O2WebImageAttributes,
O2WebFormAnalyser;
```

Class Directories

We define the **`html_header`** method in the **`Directories`** class in order to personalize the header when retrieving objects of this class.

```
method body html_header(query:string, userdata:string):bits
in class Directories
{
      o2 bits result;

      result += "<html> <header>\n";
      result += "<title> directories </title>\n";
      result += "</header><body>\n";
      result += "<center><h2>\n";
      result += "The Available Directories";
      result += "\n</h2></center>\n";
      result += "<hr>";
      return result;
};
```

Class Directory

We now add two methods to the **`Directory`** class. The first one (**`html_header`**) is used to personalize the header of objects of the **`Directory`** class.

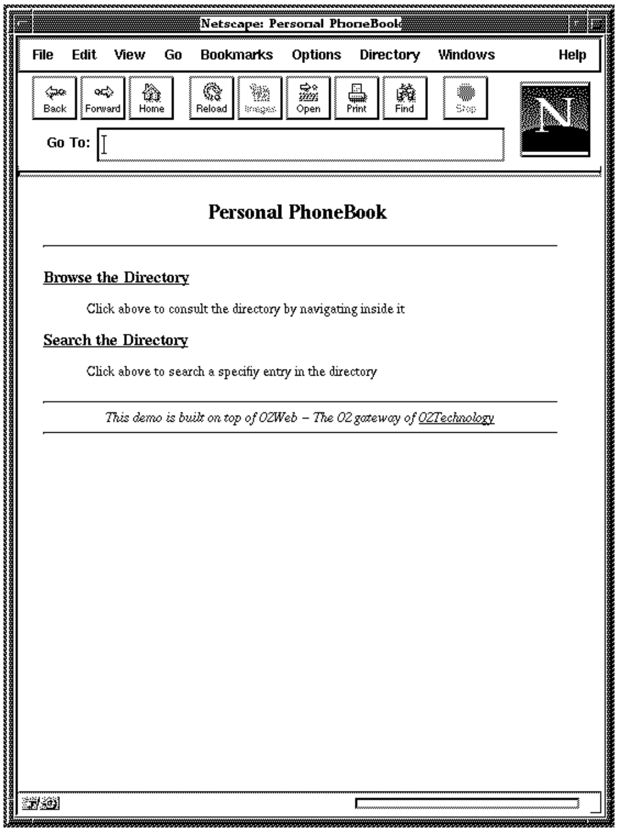
```
method body html_header(query:string, userdata:string):bits
in class Directory
{
      o2 bits result;
      result += "<html> <header>\n";
      result += "<title>" +  self->html_title + "</title>\n";
      result += "</header><body>\n";
      result += "<center><h2>\n";
      result += self->html_title;
      result += "\n</h2></center>\n";
      result += "<hr>";
      return result;
};
```

The second one (**`html_report`**) is used to personalize HTML generation. We change the standard HTML generation and give the user the choice whether to browse the directory or search for an entry in the directory.

The first choice is associated with a URL containing a query leading to the **entries** field of the class. Notice that the URL is created using the **make_url** method in the **O2WebAssistant** class. The **userdata** parameter specifies the name of the directory. This is returned to the **html_header**, **html_footer** and **html_report** methods which are called when a user clicks on the created anchor. The **userdata** parameter is detailed in the **make_url** method of the O_2WebAssistant Toolkit. This is a means of storing a context in the returned HTML and retrieving it when a user clicks on an anchor. This could have been achieved using other techniques such as hidden fields or cookies.

The second choice is associated with a URL leading to an object called **TheDirectorySearcher**. This object belongs to a new class **DirectorySearch**. This class handles the creation of an HTML form and the retrieval of user information. This class will be discussed below.

```
method body html_report(query:string, userdata:string):bits
in class Directory
{
  o2 bits result;
  o2 bits anchor;
  o2 O2WebAssistant toolbox = new O2WebAssistant;

  result += "<dl>\n";
  anchor  = toolbox->make_anchor(
       toolbox->make_url( self->get_query + "->entries", "",
                          self->html_title, 0),
       "Browse the Directory");
  result += "<dt> <h3>" + anchor +"</h3>\n";
  result += "<dd> Click above to consult the directory by
navigating inside it\n";
  anchor  = toolbox->make_anchor(
       toolbox->make_url( "TheDirectorySearcher",
                          "",self->html_title, 0),
       "Search the Directory");
  result += "<dt> <h3>" + anchor + "</h3>\n";
  result += "<dd> Click above to search a specify entry in the
directory\n";
  result += "</dl>\n";
  return result;
};
```


Netscape: Personal PhoneBook
File Edit View Go Bookmarks Options Directory Windows Help
Back Forward Home Reload Images Open Print Find Stop
Go To:

Personal PhoneBook

Browse the Directory

Click above to consult the directory by navigating inside it

Search the Directory

Click above to search a specifiy entry in the directory

This demo is built on top of O2Web – The O2 gateway of O2Technology

Figure 5.1: The customized Directory class

Class Entries

We personalized the header for an object of the class **`Entries`**. You may be wondering why we chose to define a class to manage the entries in a directory rather than defining an attribute of the **`Directory`** class to contain the list of entries. This choice was made to enable the personalization of the pages containing the list of entries.

Notice that the **`userdata`** parameter is used in the method below. It contains a string, given by the programmer in the **`html_report`** method of the **`Directory`** class when the anchor pointing to an object of the **`Entries`** class was created. This string contains the name of the directory.

```
method body html_header(query:string, userdata:string):bits
in class Entries
{
      o2 bits result;
      result += "<html> <header>\n";
      result += "<title> " + userdata + " entries </title>\n";
      result += "</header><body>\n";
      result += "<center><h2>\n";
      result += "Entries of the " + userdata + " Directory";
      result += "\n</h2></center>\n";
      result += "<hr>";
      return result;

};
```

Class Entry

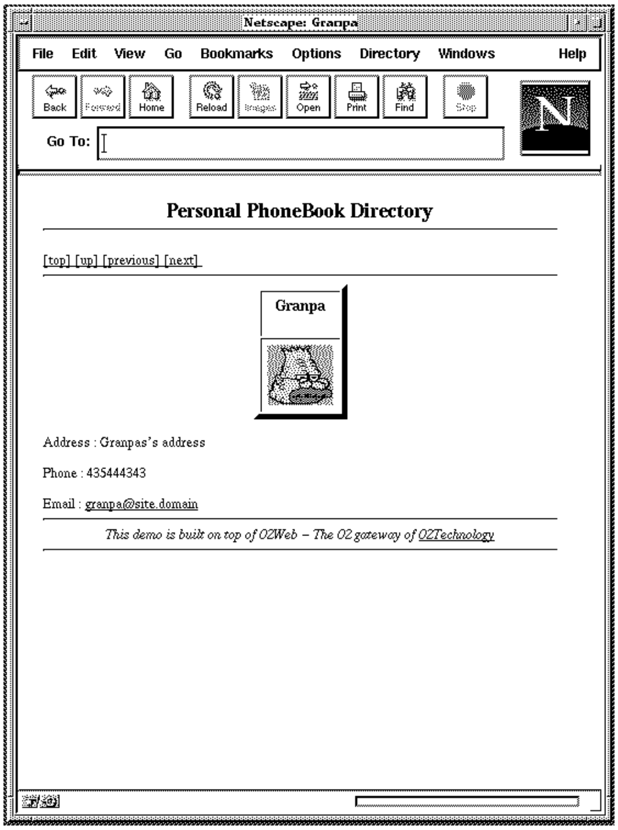
We define two methods in the **`Entry`** class. First, we add a specific header. Previously, the **`previous`**, **`next`** and **`up`** attributes of the class **`Entry`** were part of the report and appeared as anchors. We now want to remove these attributes from the report and add a navigation bar to the header defined in this class.

The navigation bar provides direct access to the previous and next entries of the current phone book as well as access to the phone book (**`up`**) and the list of phone books (**`top`**).

```
method body html_header(query:string, userdata:string):bits
in class Entry
{
  o2 bits result;
  o2 O2WebAssistant toolbox = new O2WebAssistant;
  o2 string url;

  result += "<html> <header>\n";
  result += "<title> " + self->name + "</title>\n";
  result += "</header><body>\n";
  result += "<center><h2>\n";
  result += self->up->name + " Directory";
  result += "<hr>\n";
  result += "\n</h2></center>\n";

  if (DIRECTORIES != nil) {
    url = toolbox->make_url("DIRECTORIES", "", "", 0);
    result += toolbox->make_anchor(url, "[top] ");
  }
  result += " ";
  if (self->up != nil) {
    url = toolbox->make_url(self->up->get_query, "", "", 0);
    result += toolbox->make_anchor(url, "[up] ");
  }
  result += " ";
  if (self->previous != nil) {
    url = toolbox->make_url(  self->previous->get_query, "",
                              "", 0);
    result += toolbox->make_anchor(url, "[previous] ");
  }
  result += " ";
  if (self->next != nil) {
    url = toolbox->make_url(self->next->get_query, "", "", 0);
    result += toolbox->make_anchor(url, "[next] ");
  }
  result += "<hr>";
  return result;
};
```

Now we define a specific report method that formats an entry in the directory. Note the use of the **`<table>`** tag in the code below. Tables are part of the HTML 3.0 specification and are not recognized by all browsers.

```
method body html_report(query:string, userdata:string):bits
in class Entry
{
  o2 bits result;
  o2 O2WebAssistant toolbox = new O2WebAssistant;
  o2 O2WebImageAttributes format = new O2WebImageAttributes;

  result += "<center>\n";
  result += "<table border=5 cellpadding=5>\n";
  result += (o2 bits)"<tr align=center>\n";
  result += (o2 bits)"<td>"+"<h3>"+self->name+"</h3></td>\n";
  result += "<tr align=center>\n <td>";
  result += toolbox->make_inline_image( self->get_query + "->photo",
                                       0,0,format, 0,"Entry Photo");
  result += "</td>";
  result += "</table>\n";
  result += "</center>\n";
  result += "<p>Address : " + self->address;
  result += "<p>Phone   : " + self->phone;
  result += (o2 bits)"\n<p>Email   : " +
                            "<a href=\"mailto:" +
                            self->e_mail + "\">" +
                            self->e_mail + "</a>\n";
  return result;
};
```


Netscape: Granpa
File Edit View Go Bookmarks Options Directory Windows Help
Back Forward Home Reload Images Open Print Find Stop
Go To:

Personal PhoneBook Directory

[top] [up] [previous] [next]

Granpa

Address : Granpas's address

Phone : 435444343

Email : granpa@site.domain

This demo is built on top of O2Web – The O2 gateway of O2Technology

Figure 5.1: The customized Entry class

Class DirectorySearch

We saw in the **`html_report`** method of the **`Directory`** class that the user has the choice of browsing the directory or searching for a specific entry in the directory. The following class, called **`DirectorySearch`**, displays an HTML form that enables the user to search a directory. The method **`html_report`** creates the form and the **`search`** method analyzes the user's answer. A named object, **`TheDirectorySearcher`,** is created for this class.

```
class DirectorySearch
end;
method public html_header(query:string, userdata:string):bits
  in class DirectorySearch;
method public html_report(query:string, userdata:string):bits
  in class DirectorySearch;
method public search(params:bits):bits
  in class DirectorySearch;

name TheDirectorySearcher:DirectorySearch;
```

The **html_header** method is used to personalize the header for objects of this class.

```
method body html_header(query:string, userdata:string):bits
in class DirectorySearch
{
      o2 bits result;
      result += "<html> <header>\n";
      result += "<title> " + userdata + " search </title>\n";
      result += "</header><body>\n";
      result += "<center><h2>\n";
      result += "Search the " + userdata + " Directory";
      result += "\n</h2></center>\n";
      result += "<hr>";
      return result;
};
```

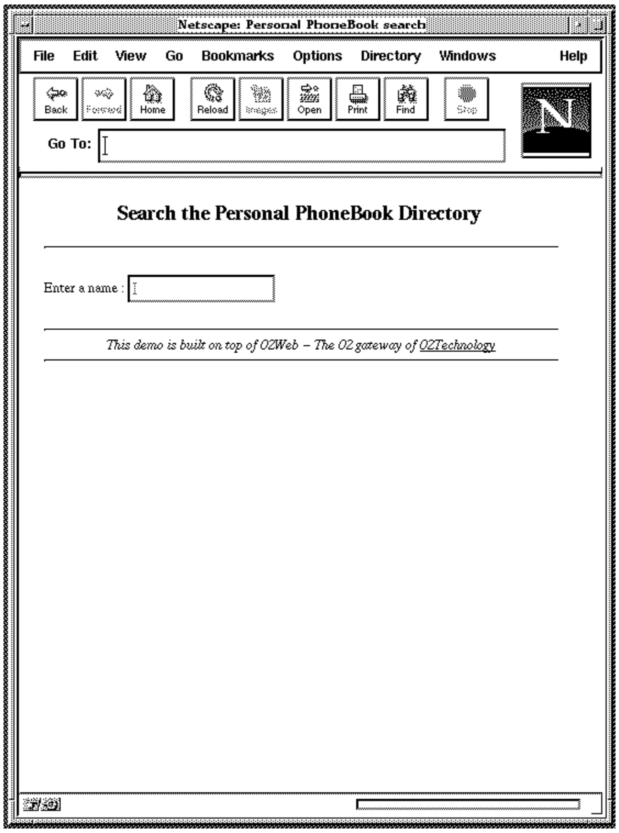
The **html_report** method creates an HTML form with two fields: the first field is used by the user to enter a name to search for in the directory; the second field is a hidden field (invisible on the screen) that retains the name of the directory, which was searched, in the HTML produced.

The action of the form is defined as:

TheDirectorySearcher->search($0)

This means that when a form is submitted, the method **search** is triggered on the **TheDirectorySearcher** root. The values of the form fields are given to the **search** method by substituting the string $0 with the result of the form.

```
method body html_report(query:string, userdata:string):bits
in class DirectorySearch
{
      o2 bits result;
      o2 O2WebAssistant toolbox = new O2WebAssistant;

      result += "<form method=post action=";
      result += toolbox->make_url(
                                "TheDirectorySearcher->search($0)",
                                "","",0);
      result += ">\n";

      result += "Enter a name : <input name=\"pattern\"
value=\"\">";
      result += "<input type=hidden name=\"directory\" value=\"" +
userdata + "\">";
      result += "</form>";
      return result;
};
```



Search the Personal PhoneBook Directory

Enter a name :

This demo is built on top of O2Web – The O2 gateway of O2Technology

Figure 5.1: An interface for searching

The **search** method retrieves the form result using the **O2WebFormAnalyser** class of the O$_2$WebAssistant Toolkit. Then it searches for the entry and displays the result. Notice, as with the **error** method of the class **O2WebInteractor**, a method called as an action of a form must return a complete HTML text, including the CGI header. This is why the **search** method creates a return value containing a call to **get_http_prolog** when the entry does not exist in the directory. When the entry is found, the result of the method is the result of the call to the **make_report** method of the **O2WebAssistant** class. To insert the CGI header, the **make_report** method is called with the **RP_WITH_HEADER** parameter.

```
method body search(params:bits):bits
in class DirectorySearch
{
#include "o2web.h"

      o2 bits result;
      o2 string name, dir_name;
      o2 O2WebAssistant toolbox = new O2WebAssistant;
      o2 O2WebFormAnalyser formtool=new O2WebFormAnalyser(params);
      o2 list(O2WebFormItem) formitems;
      o2 Directory d, directory;
      o2 Entry e, entry;

      formitems = formtool->get_values("pattern");
      if (count(formitems) > 0)
            name = (formitems[0])->get_value;

      formitems = formtool->get_values("directory");
      if (count(formitems) > 0)
            dir_name = (formitems[0])->get_value;

      for (d in *DIRECTORIES)
            if (d->html_title == dir_name) {
                  directory = d;
                  break;
            }

      if (directory!=nil) {
            for (e in *(directory->entries))
                  if (e->html_title == name) {
                        entry = e;
                        break;
                  }
      }

      if (entry == nil) {
            result += toolbox->get_http_prolog("text/html");
            result += "<h3>";
            result += "There is no entry for "+name+" in "+dir_name;
            result += "</h3>";
      } else {
            result += toolbox->make_report(
                              (int)(o2 void)entry,
                              entry->get_query, RP_DEFAULT,
RP_WITH_HEADER);
      }
      return result;
};
```

5.5 Updating the database with O2Web

There are many occasions where a programmer might wish to update the database (log a connection in the database, store the result of an HTML form, etc.).

As O_2Web runs in program mode, a method can update persistent objects of the database or create new persistent objects only if it starts a new transaction. It is the responsibility of the programmer to end the transaction after the updates in order to leave O_2Web in read-only transaction mode.

In O_2, a transaction is ended using either the **validate** instruction or the **commit** instruction. In O_2Web, a transaction can only be ended using the **validate** instruction. If a method called by O_2Web performs a **commit** instruction, the current request is discarded and the Web client receives an error message.

Suppose you want to log information concerning a connection in the database; you can define the **disconnect** method of the **O2WebInteractor** class as follows:

```
method body disconnect(report_size:integer, report_kind:string)
in class O2WebInteractor
{
      TheConnections->add(report_size, report_kind);
};
```

It is up to the programmer to start a new transaction in order to update the **TheConnections** persistent object. This must be done as follows:

```
method body add(size:integer, kind:string)
in class Connections
{
      o2 Connection connection = new Connection;
      connection->size = size;
      connection->kind = kind;

      transaction;
      self->connections += list(connection);
      validate;
};
```

Furthermore, as the method above might produce a deadlock when more than one O_2Web server is accessed at the same time to answer requests, an explicit lock must be set on the persistent object before manipulating it in order to avoid deadlocks. Thus, the method should be written as follows:

```
method body add(size:integer, kind:string)
in class Connections
{
      o2 Connection connection = new Connection;
      connection->size = size;
      connection->kind = kind;
      transaction;
      o2_lock((o2 void) self, O2_LOCK_EXCLUSIVE, O2_LOCK_WAIT);
      self->connections += list(connection);
      validate;
};
```

For further information concerning transactions and deadlocks, please refer to the O_2 manuals.

5.6 Users and Sessions

For security control, you can use predefined classes such as **`O2WebUser`** to identify each user who connects. Based on the user's identity and status, you can then programmatically control access to information on the server. You can also keep a profile on each user's preferences. Finally, you can define a class to record information about a series of user queries, essentially emulating a session, then use this information to allow or deny access to objects at the end of the series of queries.

Users are objects in the **`O2WebUser class`**, which maintains the full name, login, password, status, and groups to which the user belongs.

Users are assigned one of the following statuses:

- Authenticated: O2Web uses one of the following methods to identify the user. Within O2Web you can obtain the user's login and password in one of the following ways:
  - HTTP — A pop up window displays on the browser, requesting login and password. You can modify text appearing in this window by calling the **`set_realm`** method in the **`O2WebUserManager`** class.
  - HTML — A form requests login and password. You can use the default page, or you can define your own by defining the **`login`** method in the **`O2WebInteractor`** class.

Users are authorized for the amount of time specified in the **`set_timeout`** method in the **`O2WebUserManager`** class, after which they default to Identified status.

- Identified: Users default to this status after timing out from the Authenticated status. O_2Web uses the magic cookies method of identifying the user.
- Anonymous: O_2Web uses the magic cookies mechanism to map the user with a connection without obtaining the user's login name or password.

The **`html_check`** method controls access to objects for security purposes. This method is called before **`html_prolog`**, **`html_header`**, **`html_report`**, and **`html_footer`**. Within **`html_check`**, you can call the **`get_current_user`** method in the **`O2WebUserManager`** class to obtain information about the user, including identity and status. This class belongs to the **`O2web`** schema, which is provided with O_2Web. Based on the user's status, **`html_check`** allows or denies access to the server by returning one of the following values: **`O2WEB_ACCEPT`**, **`O2WEB_DENY`**, or **`O2WEB_CHECK`**.

Warning !

You must import the persistent root **TheO2WebUserManager** from the **o2web** schema before you can use **get_current_user** to retieve user information, as in the following example.

```
import schema o2web class O2WebUserManager, O2WebUser, O2WebGroup;
import schema o2web name TheO2WebUsermanager;

class Person type tuple (name: string, age: integer) end;
method html_check:integer in class Person;

method body html_check:integer in class Person
{
      o2WebUser user= TheO2WebUserManager->get_current_user();
      if (user->get_status == O2WEB_AUTHENTIFIED)
            return O2WEB_ACCEPT;
      if (user->get_status == O2WEB_IDENTIFIED ||
            user->get_status == O2WEB_ANONYMOUS)
            return O2WEB_CHECK;
};
```

5.7 Summary

In this section, we summarize the techniques used to create an HTML text for a query submitted to an O_2Web server.

The HTML text produced for any query is made of five parts:

1 prolog - a protocol specific text

2 header - a constant page header

3 body - the body of a report

4 footer - a constant page footer

5 epilog - a protocol specific text

All these document parts have a generic implementation that can be redefined by the programmer.

The prolog, header, footer and epilog sections can be overloaded globally in the **`O2WebInteractor`** class by means of the **`prolog`**, **`header`**, **`footer`** and **`epilog`** methods and overloaded locally in any class using the **`html_prolog`**, **`html_header`**, **`html_footer`** and **`html_epilog`** methods.

The body of a report can only be overloaded locally by means of the **`html_report`** method.

The generic implementation can also be improved by locally defining the **`get_query`** methods in your classes.

Besides HTML production personalization, the programmer can personalize the reaction to certain events that might occur. These events are a connection, a disconnection and an error. The handlers associated with these events are methods defined in the **`O2WebInteractor`** class.

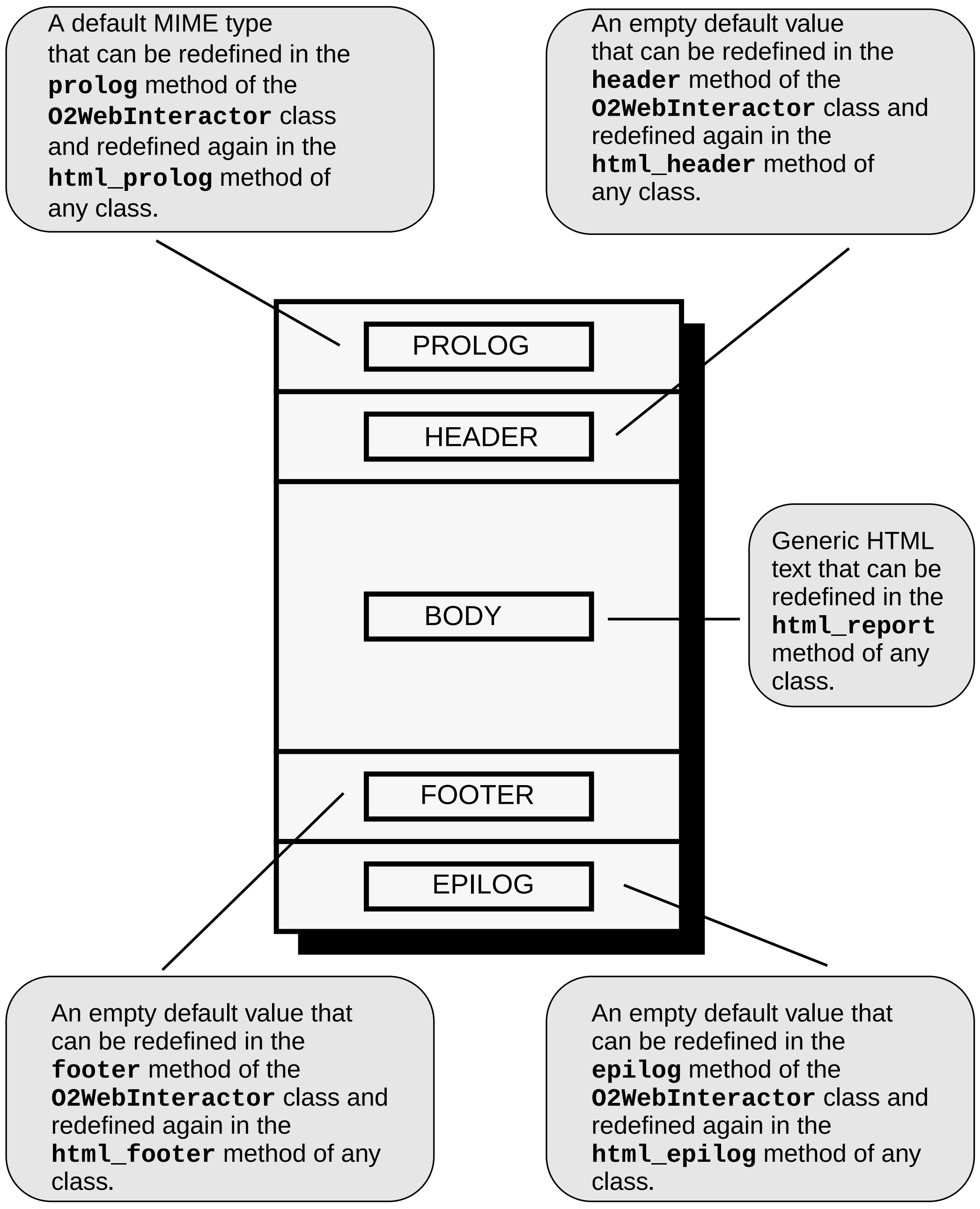

A default MIME type that can be redefined in the **prolog** method of the **O2WebInteractor** class and redefined again in the **html_prolog** method of any class.
An empty default value that can be redefined in the **header** method of the **O2WebInteractor** class and redefined again in the **html_header** method of any class.
PROLOG
HEADER
BODY
FOOTER
EPILOG
Generic HTML text that can be redefined in the **html_report** method of any class.
An empty default value that can be redefined in the **footer** method of the **O2WebInteractor** class and redefined again in the **html_footer** method of any class.
An empty default value that can be redefined in the **epilog** method of the **O2WebInteractor** class and redefined again in the **html_epilog** method of any class.

Figure 5.8: HTML production

6 O2Web Reference

This chapter provides the syntax and usage for O_2Web methods and commands.

It is divided into the following sections:

- The O2WebInteractor Class — Programmers can define some methods in this class in order to overload O_2Web behavior for all classes of a schema that do not have their own behavior. This class is also used by programmers to provide O_2Web with methods to call when some events occur (connection, disconnection, etc.).
- User-Defined Methods — Programmers can define methods for each class of a schema, to overload the default behavior or the global redefinition of the **`O2WebInteractor`** class.
- The O2WebAssistant Toolkit — This is composed of several classes and methods whose aim is to help the programmer in the process of redefining the generic HTML production.
- The O2Web Commands — This section outlines the O_2Web system commands.

6 O2Web Reference

6.1 The O2WebInteractor Class

`O2WebInteractor` is an empty class that can be created or imported from the **`o2web`** schema. Some methods can be defined in this class to globally change parts of the generic HTML production.

Other methods can be defined to handle certain events such as connection, disconnection and error.

Methods of the **`O2WebInteractor`** class are called only if a persistent root called **`TheO2WebInteractor`** is defined in this class.

The rest of this section describes the following methods:

- **`connect`**
- **`disconnect`**
- **`epilog`**
- **`error`**
- **`footer`**
- **`header`**
- **`login`**
- **`prolog`**
- **`value_report`**

connect

Summary A user-defined method called once for each connection.

Syntax `connect(query:string,userdata:string):boolean`

Arguments

`query`	The query to be submitted to the O_2Web server.
`userdata`	A string that the programmer might have inserted in the anchor that produced the query. If this is the case, this string is returned to the programmer (for more information about **`userdata`**, see the **`make_url`** method of the O_2WebAssistant Toolkit).

Description This method is called once for each connection if the **`TheO2WebInteractor`** persistent root is defined. There is no default implementation for this method. It can be defined for desired tasks such as authorization checking.

Returns A boolean.

A value of **`false`** causes O_2Web to reject the connection. O_2Web will try to trigger the **`error`** method on the **`TheO2WebInteractor`** root. The argument for this method are **`O2WEB_PROTECTION`**.

A value of **`true`** authorizes the connection.

Example In this example **`connect`** is used to check the access authorization of the users connecting to the server.

```
method body connect( query:string,
                     userdata:string):boolean
{
  o2 O2WebAssistant tool = new O2WebAssistant;
  o2 string remote_addr =
            tool->get_http_variable("REMOTE_ADDR");
  if (!is_an_authorised_address(remote_addr))
    return false;
  else return true;
};
```

disconnect

Summary	A user-defined method called once after each connection.
Syntax	**`disconnect(report_size: integer, report_kind:string)`**
Arguments	**`report_size`** The size of the generated document.
	`report_kind` The MIME type of the returned document.
Description	This method is called once after each connection if the **`TheO2WebInteractor`** persistent root is defined. There is no default implementation for this method. It can be defined for desired tasks such as statistical analysis.
Returns	None.
Example	In this example **`disconnect`** is used to log information concerning requests in the database. The method receives the size and MIME type of the returned document and some HTTP environment variables as arguments.

```
method body disconnect( report_size:integer,
                        report_kind:string)
{
  o2 O2WebAssistant tool = new O2WebAssistant;
  o2 string rem_addr =
            tool->get_http_variable("REMOTE_ADDR");
  o2 string http_meth =
            tool->get_http_variable("REQUEST_METHOD");

  TheConnections->add( report_size, report_kind,
                       rem_addr, http_meth);
};
```

As O_2Web runs in program mode, it is up to the programmer to start a new transaction in order to update persistent objects.

Furthermore, as the following **`add`** method could produce a deadlock when more than one O_2Web server is accessed at the same time to answer requests, an explicit lock is set on the persistent object before it is manipulated.

```
method body add( size:integer, kind:string, addr:string,
                 meth:string)
in class Connections
{
  #include "o2.h"
  o2 Connection connection = new Connection;
  connection->size = size;
  connection->kind = kind;
  connection->remote_addr = addr;
  connection->http_method = meth;
  transaction;
  o2_lock(    (o2 void) self, O2_LOCK_EXCLUSIVE,
              O2_LOCK_WAIT);
  self->connections += list(connection);
  validate;
```

epilog

Summary	Changes the default epilog.
Syntax	**`epilog: string`**
Arguments	None.
Description	This method is called each time HTML is generated if the **`TheO2WebInteractor`** persistent root is defined and if the **`html_epilog`** method has not been defined in the class to which belongs the object computed by the URL query. There is no default implementation for this method.
Returns	An O_2 string.

error

Summary Redefines the error message.

Syntax

```
#include "o2web.h"
error(kind:integer):bits
```

Arguments

kind	An error code.

Description This allows the programmer to redefine the error message returned to the Web client. It is called when an error occurs if the **TheO2WebInteractor** persistent root is defined. The possible values of **kind** are:

- **O2WEB_WRONG_BASE_NAME**
- **O2WEB_EMPTY_QUERY**
- **O2WEB_RUNTIME_ERROR**
- **O2WEB_COMMIT**
- **O2WEB_ABORT**
- **O2WEB_BAD_FORM**
- **O2WEB_PROTECTION**
- **O2WEB_DENY**

Some of the above errors can occur due to a problem in the HTTP configuration file.

Returns An O_2 bits value that contains the HTML text, including the CGI header, to be sent to a Web client when an error occurs.

Example When an error occurs during O_2Web activity, default error messages are sent to the Web client. The following example shows how a programmer can define his own error messages.

```
method body error(kind:integer):bits
{
#include "o2web.h"
  o2 bits error;
  o2 O2WebAssistant tool = new O2WebAssistant;

  error += tool->get_http_prolog("text/html");
  switch(error) {
    case O2WEB_WRONG_BASE_NAME:
      error += "Please retry with a valid database name.";
      break;
    case O2WEB_PROTECTION:
      error += " The requested information is protected. Sorry.";
      break;
    case O2WEB_BAD_FORM:
      error += "Sorry, an error has occurred while building an ";
      error += "answer to your form request.";
      break;
    default:
      error += "Sorry, an error has occurred. <p>";
      error += "If the error persists, contact:<br>";
      error += "<a href=\"mailto:webmaster@site.dom\">";
      error += "webmaster@site.dom </a>";
      break;
  ]
  return error;
};
```

footer

Summary Adds a constant footer to the bottom of each page.

Syntax **`footer:bits`**

Arguments None.

Description This method adds a constant footer to the bottom of each page. It is called after each time HTML is produced if the **`TheO2WebInteractor`** persistent root is defined and if a method **`html_footer`** has not been defined in the class to which belongs the object computed by the URL query.

There is no default implementation for this method.

This method is used to add constant elements to the bottom of each page returned by O_2Web.

Returns An O_2 bits value containing a piece of HTML text to be inserted at the bottom of each page.

Example A simple footer can be written as follows:

```
method body footer:bits
{
  o2 bits footer;

  footer += "<hr>";
  footer +=   "<address><center> O2 Technology
              </center></address>";
  footer += "<hr>";
  return footer;
};
```

header

Summary Adds a constant header to the top of each page.

Syntax **`header:bits`**

Arguments None.

Description This method adds a constant header to the top of each page. It is called each time HTML is produced if the **`TheO2WebInteractor`** persistent root is defined and if a method **`html_header`** has not been defined in the class to which belongs the object computed by the URL query.

There is no default implementation for this method.

This method is used to add constant elements to the top of each page returned by O_2Web.

Returns An O_2 bits value containing a piece of HTML text to be inserted at the top of each page.

Example A simple header can be written as follows:

```
method body header:bits
{
  o2 bits header;

  header += "<hr>";
  header += "<h1> My Header </h1>";
  header += "<hr>";
  return header;
};
```

login

Summary Replaces the default login page.

Syntax **`login:bits`**

Arguments None.

Description This method is called each time O_2Web needs to obtain the user's name and password when the global O_2Web protection mode is HTML. See Chapter 5 for information on protection modes. O_2Web calls this method when the **`html_check`** method returns **`O2WEB_CHECK`** because the user is not known (this is the user's first query), or because the user times out of authenticated mode.

The **`login`** method must return an HTML form requesting the user's login (in the **`login`** field) and password (in the **`passwd`** field). The method for the form must be post, and the action of the form must be <*database_name*>?**`VerifyO2WebPasswd`**.

The default method is used to produce the login page if the login method is not defined in the **`O2WebInteractor`** class or if the persistent root **`TheO2WebInteractor`** is not defined.

Returns An O_2 bits value containing a form that requests the user's login and password.

prolog

Summary	Changes the default prolog.
Syntax	**`prolog:string`**
Arguments	None.
Description	This method is called each time HTML is generated if the **`TheO2WebInteractor`** persistent root is defined and if the **`html_prolog`** method has not been defined in the class to which belongs the object computed by the URL query. The default implementation of this method returns a CGI header describing the type of the returned document.
Returns	An O_2 string that must contain a valid CGI header.

value_report

Summary	Overloads the method that responds to queries requesting values rather than objects.
Syntax	**`value_report:bits`**
Arguments	None.
Description	This method is called when a URL contains a query requesting a value rather than an object. This method ensures that the results have been generated by a custom method rather than the default method. If **`value_report`** is not defined in the **`O2WebInteractor`** class, or if the persistent root **`TheO2WebInteractor`** is not defined, the default method is used.
Returns	An O_2 bits value containing the HTML stream to present to users when a query in the URL requests a value.

6.2 User-Defined Methods

The previous section described how to globally overload part of the HTML production. This section explains how to locally redefine (for a class) the HTML generation.

A local redefinition is performed by defining methods in a class. Each defined method overloads a part of the HTML generation.

The methods that a programmer can define are:

- **`get_query`**
- **`html_check`**
- **`html_epilog`**
- **`html_footer`**
- **`html_header`**
- **`html_prolog`**
- **`html_report`**
- **`html_title`**

get_query

Summary A user-defined method that must return a valid OQL query.

Syntax **`get_query:string`**

Arguments None.

Description When using the generic mode of O_2Web, hypertext links are created to allow a user to browse the sub-objects contained in an object. These links are HTML anchors that encapsulate a URL containing a query leading to a sub-object. Such a query is generated automatically by O_2Web starting from the current query and adding to it a selection predicate or an attribute name. This can produce, after many clicks, very large queries.

This problem can be avoided if the programmer knows how to directly reach objects of a class from the persistent roots. In this case, the programmer can define the **`get_query`** method in this class. The method must return a valid OQL query, which when executed will return its receiver.

Warning !

Even if the get_query method is identical in a class and its sub-class, it must be redefined in the sub-class and modified to contain a cast to the sub-class.

Returns An O_2 bits value.

Example The string returned by the method below are used by the O_2Web automatic generation as the query to be used to create an anchor leading to an object of the **`Directory`** class.

```
method body get_query:string  in class Directory
{
  return (o2 string)
    "element(select d from d in DIRECTORIES
    where d->title = "+ "\"" + self->title + "\")";
};
```

html_check

Summary	Verifies users for access to objects.
Syntax	**`html_check:integer`**
Arguments	None.
Description	This method is called before **`html_prolog`**, **`html_header`**, **`html_report`**, and **`html_footer`** on an object. Within **`html_check`**, you can call the **`get_current_user`** method in the **`O2WebUserManager`** class to retrieve information about the current user. Using the information retrieved, including the user's status and the groups to which she belongs, you can determine whether to accept the user, returning the decision in **`O2WEB_ACCEPT`**, **`O2WEB_DENY`**, or **`O2WEB_CHECK`**.
Returns	An integer in **`O2WEB_ACCEPT`**, **`O2WEB_DENY`**, or **`O2WEB_CHECK`** that determines whether the user is allowed or denied access to the object, or whether O2Web needs to requests the user's login and password.

html_epilog

Summary Specifies the prolog for a class.

Syntax **`html_epilog(query:string,userdata:string):bits`**

Parameters

`query`	The query to be submitted to the O_2Web server.
`userdata`	A string that the programmer might have inserted in the anchor that produced the query. If this is the case, this string is returned to the programmer (for more information about **`userdata`**, see the **`make_url`** method of the O_2WebAssistant Toolkit).

Description This method is called at the end of each new HTML production if it is defined on the class to which belongs the object computed by the URL query. In that case, it overloads the eventually defined **`epilog`** method of the **`O2WebInteractor`** class. The result of this method is inserted after the text returned by an **`html_footer`** method defined in the same class or the **`footer`** method of the **`O2WebInteractor`** class.

This method is rarely useful.

Returns An O_2 bits value.

html_footer

Summary Adds class-dependent elements to the bottom of each page.

Syntax **`html_footer(query:string,userdata:string):bits`**

Arguments

`query` The OQL query that returns the receiver of the method. This can be guaranteed by O_2Web only if the **`html_report`** method is called directly by O_2Web in response to a client query. If called directly by a programmer, it is its responsibility to provide the method with a correct value for the query parameter.

`userdata` A string containing data stored in an anchor by the programmer when using the **`make_url`** method of the **`O2WebAssistant`** class. This parameter is only meaningful when **`html_report`** is called by O_2Web on the result of a client query.

Description This adds class-dependent elements to the bottom of each page resulting from a query leading to the class in which this method is defined.

This method can be defined in any class of an O_2 schema. It is called, at the end of each new HTML production, if it is defined in the class to which the object computed by the URL query belongs. In this case, it overloads the eventually defined **`footer`** method of the **`O2WebInteractor`** class.

Returns An O_2 bits value containing valid HTML.

Example A simple footer for a class can be defined as follows:

```
method body html_footer(  query:string,
                          userdata:string):bits in
class Car
{
  o2 bits footer;

  footer += "<hr>";
  footer += "The footer of class Car";
  footer += "<hr>";
  return footer;
};
```

html_header

Summary Adds class-dependent elements to the top of each page.

Syntax **`html_header(query:string,userdata:string):bits`**

Arguments

`query` The OQL query that returns the receiver of the method. This can be guaranteed by O_2Web only if the **`html_report`** method is called directly by O_2Web in response to a client query. If called directly by a programmer, it is its responsibility to provide the method with a correct value for the query parameter.

`userdata` A string containing data stored in an anchor by the programmer when using the **`make_url`** method of the **`O2WebAssistant`** class. This parameter is only meaningful when **`html_report`** is called by O_2Web on the result of a client query.

Description This method is used to add class-dependent elements to the top of each page resulting from a query leading to the class in which this method is defined.

This method can be defined in any class of an O_2 schema. It is called, at the beginning of each new HTML production, if it is defined in the class to which the object computed by the URL query belongs. In this case, it overloads the eventually defined **`header`** method of the **`O2WebInteractor`** class.

Returns An O_2 bits value containing valid HTML.

Example This example defines a simple header in a class **`Car`**.

```
method body html_header(  query:string,
                          userdata:string):bits in
class Car
{
  o2 bits header;

  header += "<hr>";
  header += "The header of class Car";
  header += "<hr>";
  return header;
};
```

html_prolog

Summary Specifies the prolog for a class.

Syntax **`html_prolog(query:string, userdata:string):bits`**

Arguments

`query`	The query to be submitted to the O_2Web server.
`userdata`	A string that the programmer might have inserted in the anchor that produced the query. If this is the case, this string is returned to the programmer (for more information about **`userdata`**, see the **`make_url`** method of the O_2WebAssistant Toolkit).

Description This method is called at the beginning of each new HTML production if it is defined in the class to which belongs the object computed by the URL query. When called, it replaces the default CGI header or the one returned by the **`prolog`** method in the **`O2WebInteractor`** class (if this has been defined).

Returns An O_2 bits value containing a valid CGI header string.

Example This example defines the **`html_prolog`** method in a class for manipulating JPEG images. This tells the HTTP server that you are returning a document containing a JPEG image to it.

```
method body html_prolog(  query:string,
                          userdata:string):bits
in class JPEG
{
  o2 bits prolog;
  o2 O2WebAssistant tool = new O2WebAssistant;

  prolog = tool->get_http_prolog("image/jpeg");
  return prolog;
};
```

html_report

Summary	Replaces the default HTML generation.
Syntax	**`html_report(query:string,userdata:string):bits`**
Arguments	**`query`** The OQL query that returns the receiver of this method. This can be guaranteed by O_2Web only if the **`html_report`** method is called directly by O_2Web in response to a client query. If called directly by a programmer, it is its responsibility to provide the method with a correct value for the query parameter.
	`userdata` A string containing data stored in an anchor by the programmer when using the **`make_url`** method of the **`O2WebAssistant`** class. This parameter is only meaningful when **`html_report`** is called by O_2Web on the result of a client query.
Description	This replaces the default HTML generation for the class in which this method is defined.
Returns	An O_2 bits value containing valid HTML.
Example	The following method shows how the body of a report can be customized by a programmer.

```
method body html_report(query:string,userdata:string):bits in
class Person
{
o2 bits result;
o2 O2WebAssistant toolbox = new O2WebAssistant;
o2 O2WebImageAttributes format = new O2WebImageAttributes;

result += "<center>\n";
result += "<table border=5 cellpadding=5>\n";
result += (o2 bits)"<tr align=center>\n";
result += (o2 bits)"<td>"+"<h3>"+self->name+"</h3></td>\n";
result += "<tr align=center>\n <td>";
result += toolbox->make_inline_image(
query + "->photo",
0,0,format, 0,"Entry Photo");
result += "</td>";
result += "</table>\n";
result += "</center>\n";
result += "<p>Address : " + self->address;
result += "<p>Phone   : " + self->phone;
result += (o2 bits)"\n<p>Email   : " +
"<a href=\"mailto:" +
self->e_mail + "\">" +
self->e_mail + "</a>\n";
return result;
};
```

html_title

Summary Returns the text that will appear (in the generic mode) when the receiver is a sub-object.

Syntax `html_title:string`

Arguments None.

Description When using the generic mode of Web or the make_report method of the O2WebAssistant class, a sub-object is represented by an anchors on which users must click to enter the sub-object.

This method returns the text of the anchor. If this text has not been defined, the name of the class is used as the anchor text.

Returns An O_2 string.

Example

```
method body html_title:string in class Directory
{
  self->name;
};
```

6.3 The O2WebAssistant Toolkit

The O_2WebAssistant Toolkit is a set of classes designed to help programmers in the process of building a World Wide Web service on top of O_2.

The classes belonging to the O_2WebAssistant Toolkit are:

- **`The O2WebAssistant Class`**
- **`The O2WebFormAnalyser Class`**
- **`The O2WebFormItem Class`**
- **`The O2WebImageAttributes Class`**
- **`The O2WebImageInliner Class`**
- **`The O2WebUser Class`**
- **`The O2WebUserManager Class`**
- **`The O2WebGroup Class`**

To access the services of this class you must import them from the **`o2web`** schema.

The O2WebAssistant Class

This class is a general purpose class that contains methods used by programmers to perform various actions (retrieving a CGI variable value, creating an anchor, calling the generic O_2Web HTML generation, etc.).

This subsection presents the following methods:

- `get_http_prolog`
- `get_http_variable`
- `make_anchor`
- `make_index`
- `make_inline_image`
- `make_report`
- `make_url`

get_http_prolog

Summary	Builds the proper CGI header for a MIME Type.
Syntax	**`get_http_prolog(MimeType:string):bits`**
Arguments	**`MimeType`** The MIME type.
Description	This builds the proper CGI header for a MIME Type.
Returns	An O_2 bits value.
Example	Suppose you have defined a class to manage JPEG images. The following **`html_prolog`** method tells your HTTP server that you are returning a document containing a JPEG image to it.

```
method html_prolog:string in class JPEG
{
  o2 O2WebAssistant toolbox = new O2WebAssistant;
  returns toolbox->get_http_prolog("image/jpeg");
};
```

get_http_variable

Summary Retrieves the environment variable values of a HTTP server.

Syntax **`get_http_variable(name:string):string`**

Description This retrieves the values of the environment variables given by the HTTP server to the CGI script. The possible variable names are:

- **`SERVER_NAME`**
- **`SERVER_PORT`**
- **`SERVER_PROTOCOLE`**
- **`SERVER_SOFTWARE`**
- **`REQUEST_METHOD`**
- **`PATH_INFO`**
- **`PATH_TRANSLATED`**
- **`QUERY_STRING`**
- **`SCRIPT_NAME`**
- **`CONTENT_TYPE`**
- **`CONTENT_LENGTH`**
- **`HTTP_ACCEPT`**
- **`HTTP_USER_AGENT`**
- **`AUTH_TYPE`**
- **`REMOTE_HOST`**
- **`REMOTE_ADDR`**
- **`REMOTE_USER`**
- **`REMOTE_IDENT`**

For further information concerning these variables, consult the documentation for your HTTP server.

Returns An O_2 string.

Example This example shows how to retrieve the values of the HTTP server environment variables **`REMOTE_ADDR`** and **`REQUEST_METHOD`**.

```
method body disconnect( report_size:integer,
                        report_kind:string)
{
  o2 O2WebAssistant toolbox = new O2WebAssistant;
  o2 string remote_addr =
         toolbox->get_http_variable("REMOTE_ADDR");
  o2 string http_method =
         toolbox->get_http_variable("REQUEST_METHOD");

  Connections->add(  report_size, report_kind,
                     remote_addr,http_method);
};
```

make_anchor

Summary	Creates a bits value containing the definition of an HTML anchor.
Syntax	**`make_anchor(url:string,content:string):bits`**
Arguments	**`url`** The URL of the resource.
	`content` A string that will appear in the generated anchor.
Description	This creates a bits value containing the definition of an HTML anchor.
Returns	An O_2 bits value.
Example	In this example the **`make_anchor`** method creates an anchor from a URL and the string **`manufacturer`**.

```
method body html_report(  query:string,
                          userdata:string) in class Car
{
  o2 O2WebAssistant toolbox = new O2WebAssistant;
  o2 string url;
  o2 string result;
  ...
  url += toolbox->make_url(
         query+"->manufacturer",
         "",
         self->name,
         1234);
  result += "Browse the ";
  result += toolbox->make_anchor(url, "manufacturer");
  result += " of this car.";
  ...
};
```

make_index

Summary Creates a string that contains an HTML index.

Syntax `make_index(name:string,content:string):bits`

Arguments	`name`	The name of the created string.
	`content`	The content of the created string.

Description This creates a string that contains an HTML index. This index can be referred to when creating a URL in order to scroll through the retrieved document until the index becomes visible.

Returns An O_2 string.

Example

```
method body html_report(  query:string,
                          userdata:string) in class Car
{
  o2 O2WebAssistant toolbox = new O2WebAssistant;
  o2 bits result;
  ...
  result += " ....The ";
  result += toolbox->make_index("idx_manu",
                                 "manufacturer";
  result += " of this car ....";
  ...
};
```

The call to the **`make_index`** method inserts an invisible marker in the text. This marker can be used by an anchor going to that document to make it scroll through the document until the marker becomes visible.

Building an anchor that goes to a marker inside a document is achieved by specifying the marker name in the **`make_url`** method. For instance, if a document contains an anchor built using a URL as follows:

```
url += tool->make_url(query, "idx_manu", "", 0);
```

Clicking the anchor retrieves the document built by the method above and scrolls through it until the index become visible.

make_inline_image

Summary Creates an inline image.

Syntax

```
make_inline_image( query:string,
                   width:integer, height:integer,
                   format:O2WebImageAttributes,
                   key:integer, alphalabel:string):bits
```

Arguments

`query`	A query leading to an image.
`width`	The width of an image. If this is not equal to 0, the value is used by some browsers to reserve space for the image in order to continue displaying the text of the received document before loading the inline image.
`height`	The height of an image. If its value is not equal to 0, it is used by some browsers to reserve space for the image in order to continue displaying the text of the received document before loading the inline image.
`format`	An object of the **`O2WebImageAttributes`** class containing directives to change the attributes (borders, alignments, clickable, etc.) of the image. Giving nil for **`format`** indicates that the image is not clickable, the borders are null, and the alignment of the image is the default used by the browser.
`key`	A short integer that encodes the query. It is inserted into the bits result of this method. A value of 0 means no encoding.
`alphalabel`	A string that is displayed instead of the inline image by text-oriented browsers or when a browser is configured to load images only on demand.

Description This creates an inline image.

Returns An O_2 bits value.

Example The following example demonstrates the creation of an inline image.

```
method body html_report(query:string,
       userdata:string):bits in class Car
{
     o2 bits result;
     o2 O2WebAssistant toolbox = new O2WebAssistant;
     o2 O2WebImageAttributes format =
       new O2WebImageAttributes;

     format->set_border(3);
     result += "<center>\n";
     result +=
       toolbox->make_inline_image( query + "->photo", 0,
                                   0, format, 0,
                                   "Car Image");
     result += "</center>";
     ...
  return result;
};
```

make_report

Summary Produces an HTML output of a complex O_2 value.

Syntax

```
#include "o2web.h"
make_report(obj:integer,
            query:string,
            generic: integer,
            header:integer,
            userdata:string):bits
```

Parameters

`obj`	The value to be printed. It can be an object, a tuple, a collection, a string or a byte (but not an integer, a char, a boolean or a real).
`query`	A query leading to the value **`obj`**.
`generic`	This indicates whether **`make_report`** uses user-defined methods (**`RP_DEFAULT`**) or generic methods (**`RP_GENERIC`**) to build the report.
`header`	This indicates whether O_2Web adds a CGI header (**`RP_WITH_HEADER`**) or not (**`RP_NO_HEADER`**).
`userdata`	A string to be returned to the programmer in the **`html_header`**, **`html_footer`**, **`html_report`** methods when the user clicks on an anchor associated with this URL

Description This produces an HTML output of a complex O_2 value. It embeds a generated report within a report.

Returns An O_2 bits value.

Example The following example creates a report that includes the name and the price of the car and then inserts a report of the description of the car.

```
method body
html_report(query:string,userdata:string):bits in
class Car
{
  #include "o2web.h"
    o2 bits result;
    o2 O2WebAssistant toolbox = new O2WebAssistant;

    result += "<h3>";
    result += self->name + "<br>";
    result += self->price + "<br>";
    result += "</h3>";
    result += toolbox->make_report(
                          self->description,
                          query + "->description",
                          RP_DEFAULT,
                          RP_NO_HEADER, "");
  return result;
};
```

make_url

Summary Creates a formatted URL.

Syntax

```
make_url(query:string,index:string,
         user_data:string,key:integer):string
```

Arguments

query	A query.
index	A string referring to the index name of the returned document.
user_data	A string to be returned to the programmer in the **html_header**, **html_footer** and **html_report** methods when the user clicks on an anchor associated with the URL.
key	A short integer used to encode the query inserted in the URL. A value of 0 means no encoding.

Description This creates a formatted URL leading to the object result of **query**.

Returns An O_2 string.

Example In this example a call to the **make_url** method will create a string containing the URL leading to the **manufacturer** attribute of the receiver of this method. The receiver name is given in the **userdata** attribute of this method. Furthermore, the content of the URL is encoded using the 1234 key.

```
method body html_report(  query:string,
                          userdata:string) in class Car
{
  o2 O2WebAssistant tool = new O2WebAssistant;
  o2 string url;
  ...
  url = toolbox->make_url(
       query+"->manufacturer",
       "",
       self->name,
       1234);
  ...
};
```

The O2WebFormAnalyser Class

This class helps programmers to decode HTML form results. It provides a set of methods that retrieve the keywords in a form, the number of keywords, the values retrieved for the keywords, etc.,. It is used in conjunction with the **`O2WebFormItem`** class.

The following public methods are defined:

- **`get_all_values`**
- **`get_keywords`**
- **`get_nb_values`**
- **`get_nth_value`**
- **`get_raw_data`**
- **`get_unique_keywords`**
- **`get_values`**
- **`init`**
- **`is_decoded`**

get_all_values

Summary	Retrieves a list of all the values in an HTML form.
Syntax	**`get_all_values:list(O2WebFormItem)`**
Arguments	None.
Description	This returns a list of all the values in an HTML form. When a multiple selection list is used in a form, this method returns the multiple items as different elements in the list. Each element in the list is an object of the **`O2WebFormItem`** class that contains detailed information about a single item in the form.
Returns	A list of objects from the **`O2WebFormItem`** class.

get_keywords

Summary	Returns all the keywords retrieved by an HTML form.
Syntax	`get_keywords:list(string)`
Arguments	None.
Description	This returns all the keywords retrieved by an HTML form. The same keyword may appear more than once if multiple values have been retrieved for a keyword.
Returns	A list of keywords.

get_nb_values

Summary	Returns the number of values retrieved for a particular keyword.
Syntax	`get_nb_values(name:string):integer`
Arguments	**name** A keyword.
Description	This returns the number of values retrieved for the keyword **name**.
Returns	An integer representing the number of values.

get_nth_value

Summary	Retrieves the nth value in an HTML form concerning a specific attribute.	
Syntax	**`get_nth_value(name:string, i:integer):O2WebFormItem`**	
Arguments	**name**	A string.
	`i`	An integer.

Description This returns the **`i`**th value in an HTML form concerning the **`name`** attribute. This method is only meaningful for forms containing items with potential multiple values (multiple selection list).

The number of values for an item can be retrieved using the **`get_nb_values`** method.

Returns An object of the **`O2WebFormItem`** class.

get_raw_data

Summary	Returns the form content as a raw byte string.
Syntax	**`get_raw_data:bits`**
Arguments	None.
Description	This returns the initial data retrieved from a form. It is used when O_2Web failed to decode the form.
Returns	An O_2 bits value.

get_unique_keywords

Summary	Returns all the unique keywords retrieved by an HTML form.
Syntax	`get_unique_keywords:list(string)`
Arguments	None.
Description	This returns all the unique keywords retrieved by an HTML form. Even if multiple values have been retrieved for a keyword, a keyword only appears once in the returned list.
Returns	A list of keywords.

get_values

Summary	Retrieves a list of values in an HTML form concerning a specific attribute.
Syntax	**`get_values(name:string):list(O2WebFormItem)`**
Arguments	**name** A string.
Description	This returns a list of all the values in an HTML form concerning the **`name`** attribute. When a multiple selection list is used in a form, this method returns the multiple items as different elements in the list. Each element in the list is an object of the **`O2WebFormItem`** class that contains detailed information about a single item in the form.
Returns	A list of objects from the **`O2WebFormItem`** class.

init

Summary	init constructor.
Syntax	**`init(params:string)`**
Arguments	**`params`** A string retrieved from an HTML form.
Description	This takes the string retrieved from an HTML form as its input and initializes the new object.
Returns	None.

is_decoded

Summary Establishes whether the data retrieved from an HTML form is decoded.

Syntax **`is_decoded:boolean`**

Arguments None.

Description This establishes whether the data retrieved from an HTML form has been decoded by O_2Web. O_2Web can decode HTML forms that return two kinds of data:

(1) **`application/x-www-form-urlencoded`**

(2) **`multipart/form-data`**

O_2Web retrieves the MIME type of the form data using the **`CONTENT_TYPE`** CGI variable.

(1) is usually used by all web browsers.

(2) recently introduced by Netscape, permits the retrieval of entire files from a client. Its specification conforms to RFC 1867.

If O_2Web is unable to decode data, you can get the data and decode it yourself using the **`get_raw_data`** method. This method returns the initially retrieved data.

Returns A boolean.

A value of **`true`** indicates the data has been decoded.

A value of **`false`** indicates the data has not been decoded.

6.4 The O2WebFormItem Class

The objects of this class are returned by methods of the **`O2WebFormAnalyser`** class.

The following public methods are defined:

- **`get_file`**
- **`get_name`**
- **`get_type`**
- **`get_value`**

get_file

Summary	Ascertains whether the value retrieved is the contents of a file.
Syntax	**`get_file:string`**
Arguments	None.
Description	This ascertains whether the value retrieved is the contents of a file. It is only meaningful when the form data was posted with **`multipart/form-data`** encoding.
Returns	A string containing the file name from which the value is obtained or an empty string if the value does not come from a file.

get_name

Summary	Returns the keyword for an item retrieved from an HTML form.
Syntax	**`get_name:string`**
Arguments	None.
Description	Returns the keyword for an item retrieved from HTML form.
Returns	A string containing the keyword.

get_type

Summary	Returns the type of an item retrieved from an HTML form.
Syntax	**`get_type:string`**
Arguments	None.
Description	This returns the type of an item retrieved from an HTML form. It contains the MIME type of the retrieved item (text/html etc.).
Returns	An O_2 string.

get_value

Summary	Returns the value of an item retrieved from an HTML form.
Syntax	**`get_value:bits`**
Arguments	None.
Description	This returns the value of an item retrieved from an HTML form.
Returns	An O_2 bits value.

The O2WebImageAttributes Class

This class is used to specify the attributes of an image. It must be used in conjunction with the **`make_inline_image`** method of the **`O2WebAssistant`** class or with the **`O2WebImageInliner`** class.

The following public methods are defined:

- **`set_align`**
- **`set_border`**
- **`set_clickable`**
- **`set_hspace`**
- **`set_vspace`**

set_align

Summary	Specifies the way an image is aligned with text.
Syntax	**`set_align(s:string)`**
Parameters	**s** A string value specifying the type of alignment to be used.
Description	This specifies the way an image is aligned with the current line of text. Some values are always valid whatever the browser you are using. These are **`bottom`**, **`top`**, and **`middle`**. Other values can be used but are only recognized by certain browsers such as Netscape Navigator©. These values are **`left`**, **`right`**, **`texttop`**, **`absmiddle`**, **`baseline`**, and **`absbottom`**. These relate to either floating images (**`left`**, **`right`**) or the implementation of Netscape inline images.
Returns	None.

set_border

Summary	Sets the thickness of the border around an image.	
Syntax	`set_border(b:integer)`	
Parameters	**b**	An integer indicating the thickness of the border.
Description	This sets the thickness of the border around an image.	
Returns	None.	

set_clickable

Summary	Specifies that an inline image is an imagemap.
Syntax	`set_clickable`
Parameters	None.
Description	Specifies that an inline image is an imagemap.
Returns	None.

set_hspace

Summary Sets the space to be left to the left and right of a floating image.

Syntax `set_hspace(h:integer)`

Parameters **h** An Integer indicating the space to be left to the left and right of an image.

Description This sets the space to be left between the left and right of a floating image and the text wrapped around it.

Returns None.

set_vspace

Summary	Sets the space to be left at the top and bottom of a floating image.
Syntax	`set_vspace(v:integer)`
Parameters	**v** An integer indicating the space to be left at the top and bottom of an image.
Description	This sets the space to be left between the top and bottom of a floating image and the text wrapped around it.
Returns	None.

The O2WebImageInliner Class

This class is used by the **`make_inline_image`** method of the **`O2WebAssistant`** class. It is used to generate inline images.

The following public methods are defined:

- **`set_format`**
- **`set_height`**
- **`set_key`**
- **`set_label`**
- **`set_query`**
- **`get_report`**
- **`set_width`**

set_format

Summary	Specifies the format of an inline image.	
Syntax	`set_format(f:O2WebImageAttributes)`	
Arguments	**`f`**	An object of the **`O2WebImageAttributes`** class indicating format for an inline image. Specify nil for **`f`** to format the image as follows: - the image is not clickable - no borders are drawn - the browser's default alignment is used
Description	Specifies the format of an inline image.	
Returns	None.	

set_height

Summary	Sets the height of an inline image.
Syntax	`set_height(h:integer)`
Parameters	**h** An integer indicating height of an inline image.
Description	Sets the height of an inline image. Used by some browsers to reserve space for an image. Also enables browsers to continue displaying text before an image has been completely read.
Returns	None.

set_key

Summary	Sets the key to be used to encode a query.	
Syntax	`set_key(k:integer)`	
Arguments	**k**	An integer containing the value of the key. 0 indicates no encoding.
Description	Sets the key to be used to encode a query to be inserted in the bits result of this method.	
Returns	None.	

set_label

Summary	Specifies an alpha-numerical label to be used instead of an inline image.	
Syntax	`set_label(s:string)`	
Parameters	**s**	A string containing the alphanumeric label used in place of an inline image.
Description	Specifies an alphanumeric label used in place of an inline image for browsers that do not support images or are configured to load images only on demand.	
Returns	None.	

set_query

Summary	Sets the query resulting in an image to the **`ImageInliner`** class.
Syntax	**`set_query(s:string)`**
Arguments	**s** A string that is the query.
Description	This sets the query resulting in an image to the **`ImageInliner`** class.
Returns	None.

get_report

Summary	Builds an inline image.
Syntax	`get_report:bits`
Arguments	None.
Description	Returns the HTML text that builds the image.
Returns	An O_2 bits value that builds the image.

set_width

Summary	Sets the width of an inline image.	
Syntax	`set_width(w:integer)`	
Parameters	`w`	An integer indicating the width of an inline image.
Description	Sets the width of an inline image. Used by some browsers to reserve space for an image. It also allows browsers to continue to display text before an image is completely read.	
Returns	None.	

The O2WebUser Class

This class is used to represent O_2 Web users. A user has a full name, a login, a password, and is member of one or more groups. Two objects can be associated with each user. The associated objects One associated object, called the user context, persists in the database with the user. It can be any kind of object used to store information such as the user's profile or preferences. This object's relationship with the user is established using the **`set_user_context`** method. The other associated object, called the session context, is also an object of any kind and can be used by the programmer to store information about the user's session. The relationship with the user is established using the **`set_session_context`** method. This object is removed when the session has expired.

The methods in this class include the following:

- **`set_full_name`**
- **`get_full_name`**
- **`set_login`**
- **`get_login`**
- **`set_passwd`**
- **`get_user_context`**
- **`get_session_context`**
- **`set_user_context`**
- **`set_session_context`**
- **`get_status`**
- **`add_in_group`**
- **`remove_from_group`**
- **`get_groups`**

set_full_name

Summary	Assigns the user’s full name.	
Syntax	**`set_full_name(name: string)`**	
Arguments	**`name`**	A string containing the user’s full name. An instance of the **`O2WebUser`** class.
Description	Assigns the user’s full name.	
Returns	None.	

get_full_name

Summary	Retrieves the user’s full name.
Syntax	**`get_full_name: string`**
Arguments	None.
Description	Retrieves the user’s full name.
Returns	A string containing the user’ full name.

set_login

Summary	Assigns a user’s login ID.
Syntax	`set_login(login : string)`
Arguments	`login` A string indicating the user’s login ID*.*
Description	Assigns a user’s login ID.
Returns	None.

get_login

Summary	Retrieves the user’s login ID*.*
Syntax	**`get_login: string`**
Arguments	None.
Description	Retrieves the user’s login ID*.*
Returns	A string containing he user’s login ID.

set_passwd

Summary	Assigns a user’s password.	
Syntax	`set_passwd(passwd : string)`	
Arguments	`passwd`	A string containing the user’s password.
Description	Assigns a user’s password.	
Returns	None.	

get_user_context

Summary	Retrieves user information such as profile or preferences stored in the user context object.
Syntax	**`get_user_context: Object`**
Arguments	None.
Description	Retrieves user information such as profile or preferences stored in the user context object by the **`set_user_context`** method.
Returns	The user context object.

get_session_context

Summary	Retrieves session information stored in the session context object.
Syntax	**`get_session_context: Object`**
Arguments	None.
Description	Retrieves session information stored in the session context object.
Returns	The session context object.

set_user_context

Summary	Assigns user information such as ***???*** to the user context object.
Syntax	**`set_user_context(context: Object)`**
Arguments	The user context object.
Description	Assigns user information to the user context object.
Returns	None.

set_session_context

Summary	Assigns information to the session context object.
Syntax	**`set_session_context(context: Object)`**
Arguments	The session context object.
Description	Assigns information to the session context object.
Returns	None.

get_status

Summary	Retrieves a user’s global protection status.
Syntax	**`get_status: integer`**
Arguments	None.
Description	Retrieves a user’s global protection status.
Returns	An integer indicating the user’s status, stored in one of the following variables: **`O2WEB_AUTHENTICATED`** — authenticated **`O2WEB_IDENTIFIED`** — identified **`O2WEB_ANONYMOUS`** — anonymous See Chapter 5 for an explanation of status.

add_in_group

Summary	Adds the user to a group.
Syntax	**`add_in_group(group: O2WebGroup):integer`**
Arguments	**`group`** The group to add the user to.
Description	Add the user to the group given as parameter.
Returns	An integer 0 — The user is added. -1 — The user is already a member of the group.

remove_from_group

Summary	Remove the user from a group.
Syntax	**`remove_from_group(group: O2WebGroup):integer`**
Arguments	**`group`** The group to remove the user from. An instance of the **`O2WebGroup`** class.
Description	Remove the user from the specified group.
Returns	An integer 0 — The user is removed. -1 — The user is not a member of the group.

get_groups

Summary	Retrieves the set of groups the user belongs to.
Syntax	**`get_groups: set(O2WebGroup)`**
Arguments	None.
Description	Returns a set of the groups the user belongs to.
Returns	A set of objects of the class **`O2WebGroup.`**

The O2WebUserManager Class

This class is the primary means of controlling user's access to objects. It must be imported (from the **o2web** schema) before it can be used in another schema. Furthermore, a root of persistence, called **TheO2WebInteractor**, must also be imported from the **o2web** schema for the users and sessions mechanisms of O_2Web to work properly.

The methods in this class include the following:

- **get_current_user**
- **add_regular_user**
- **get_users**
- **get_user_by_fullname**
- **get_user_by_login**
- **remove_user**
- **add_group**
- **get_groups**
- **get_group**
- **remove_group**
- **set_time_out**
- **set_anonymous_life**
- **set_realm**

get_current_user

Summary	Retrieve the current remote user.
Syntax	**`get_current_user: O2WebUser`**
Arguments	None
Description	Returns information about the current remote user. If O_2Web is not able to find the remote user, a new user is created. It is assigned a global protection status of anonymous and returned back to the caller.
Returns	An instance of the **`O2WebUser`** Class.

add_regular_user

Summary	Add a new user.
Syntax	**`add_regular_user(fullName:string, login:string, passwd:string):integer`**

Arguments

`fullName`	A string containing the user’s full name.
`login`	A string containing the user’s login ID.
`passwd`	A string containing the user’s password.

Description Add a new user.

Returns An integer
-1 — the user already exists
0 — the user is added

get_users

Summary	Retrieve a set of all users.
Syntax	**`get_users: set (O2WebUser)`**
Arguments	None.
Description	Retrieve all users.
Returns	A set of user objects of the **`O2WebUser`** class.

get_user_by_fullname

Summary	Retrieve information about a specific user by submitting a full name.
Syntax	**`get_user_by_fullname(fullName: string): O2WebUser`**
Arguments	**`fullName`** A string containing the user’s full name.
Description	Retrieve information about a specific user. The full name entered as an argument must match the full name assigned when the user was created.
Returns	Returns nil if no user with a matching full name is found. Returns the user, an instance of the **`O2WebUser`** class, if found.

get_user_by_login

Summary	Retrieve a specific user by entering a login ID.
Syntax	**`get_user_by_login(login: string): O2WebUser`**
Arguments	**`login`** A string containing the user’s login ID.
Description	Retrieve a specific user by entering a login ID.
Returns	Returns nil if no user with a matching full name is found. Returns the user, an instance of the **`O2WebUser`** class, if found.

remove_user

Summary	Remove a user from the set of users managed by O_2Web.
Syntax	**`remove_user(user: O2WebUser)`**
Arguments	**`user`** An instance of the **`O2WebUser`** class**.**
Description	Remove a user from the set of users managed by O_2Web.
Returns	None.

add_group

Summary	Create a new group.
Syntax	**`add_group(name: string):integer`**
Arguments	**name** A string containing the name of the new group.
Description	Create a new group. If a group with this name already exists, -1 is returned otherwise 0 is returned
Returns	An integer. 0 — The new group has been created. -1 — A group of this name already exists. The new group is not created

get_groups

Summary	Retrieve a set of all groups.
Syntax	**`get_groups: set(O2WebGroup)`**
Arguments	None.
Description	Builds a set of all existing groups.
Returns	A set of group objects from the **`O2WebGroup`** class.

get_group

Summary	Retrieve a group.
Syntax	`get_group(name: string): O2WebGroup`
Arguments	**`name`** A string containing the name of the group.
Description	Retrieve the named group.
Returns	An group object of the **`O2WebGroup`** class.

remove_group

Summary	Remove an existing group.	
Syntax	**`remove_group(group: O2WebGroup)`**	
Arguments	**group**	An instance of the O2WebGroup class.
Description	Remove a group from the list of groups managed by O2Web.	
Returns	None.	

set_time_out

Summary	Specify the amount of time before a user session times out.
Syntax	**`set_time_out(seconds: integer)`**
Arguments	**`seconds`** An integer indicating number of seconds.
Description	Specify the amount of time a user session is retained when no input is received from the user. After this time elapses, O2Web can potentially remove the user session. After a user's session expires, the user’s global control status is no longer authenticated. To authenticate the user again, return **`O2WEB_CHECK`** as the result of the **`html_check`** method.
Returns	None.

set_anonymous_life

Summary	Specify the amount of time before an anonymous user is removed.
Syntax	**`set_anonymous_life(seconds: integer)`**
Arguments	**`seconds`** An integer indicating number of seconds.
Description	Set the time an anonymous user is retained after the user’s last connection with O_2Web. After this time has expired , O_2Web removes the anonymous user from the set of users managed by o_2Web.
Returns	None.

set_realm

Summary	Display a window that requests the user's login.
Syntax	**`set_realm(realm: string)`**
Arguments	**`realm`** A string that contains a request for a user's login and password.
Description	Sets a string that displays in a pop up window prompting the user for login and password when the O_2Web protection mode is **`O2WEB_HTTP`**.
Returns	None.

The O2WebGroup Class

This class groups users. A group has a name and a set of related users. A user can be a member of several groups. Some methods are provided to retrieve a member of a group, to add or remove a user from a group, and to determine if a user is a member of a group.

The methods in this class include the following:

- **`set_name`**
- **`get_name`**
- **`add_user`**
- **`remove_user`**
- **`get_users`**
- **`is_member`**

set_name

Summary	Change the name of the group.	
Syntax	`set_name(name: string)`	
Arguments	**name**	A string that is the new name flor the group.
Description	Change the name of the group.	
Returns	None.	

get_name

Summary	Retrieves the name of the group.
Syntax	**`get_name:string`**
Arguments	None.
Description	Retrieves the name of the group.
Returns	Returns a string containing the group name.

add_user

Summary	Adds a user to a group.
Syntax	**`add_user(user:O2WebUser):integer`**
Arguments	**`user`** An instance of the **`O2WebUser`** Class.
Description	Adds a user to a group. The user's status determines whether the user may be added. See Chapter 4 for information on user status.
Returns	An integer. 0 — the user is added to the group -1 — The user is not added to the group.

remove_user

Summary	Removes a user from a group.	
Syntax	**`remove_user(user:O2WebUser):integer`**	
Arguments	**`user`**	An instance of the **`O2WebUser`** Class. The user you want to remove from the group.
Description	Removes a user from a group.	
Returns	An integer. 0 — The user is removed from the group. -1 — The user is not removed from the group.	

get_users

Summary	Retrieve a set of all users in the group.
Syntax	**`get_users: set (O2WebUser)`**
Arguments	None.
Description	Retrieve a set of all users in the group.
Returns	A set of user objects of the **`O2WebUser`** class.

is_member

Summary	Determine if a user is a member of the current group.
Syntax	**`is_member(user:O2WebUser):boolean`**
Arguments	None.
Description	Determine if a user is a member of the current group.
Returns	A boolean. **`true`** — The user is a member of this group. **`false`** — The user is not a member of this group.

6.5 The O2Web Commands

This section outlines the following O_2Web system commands:

The programs called by these commands can be found in the **`bin`** subdirectory of the O_2 installation directory.

The commands are:

- **`o2open_dispatcher`**
- **`o2web_gateway`**
- **`o2web_server`**

o2open_dispatcher

Summary Starts an O_2Web dispatcher.

Syntax **`o2open_dispatcher [-v]`**

Description This command starts a new O_2Web dispatcher. An O_2Web dispatcher registers all the O_2Web servers running on a LAN and is queried by the O_2Web gateway to get the address of a server able to answer an OQL query.

When choosing an O_2Web server to answer a gateway query, the dispatcher uses heuristics. A score is computed for each server running and the server with the best score is returned to the gateway. The following elements enter into the computation of the score:

- a server is running on the same host as the gateway.
- a server is already connected to the database to which the query is asked.
- the current load of each server (the number of queries treated).

Options

`-v`	display additional information on the **`o2open_dispatcher`** activity.

Files **`/etc/services`** (UNIX) or **`$WINDIR\system32\drivers\etc\services`** (Windows NT)

a file containing the port number and the protocol used by other programs to access the **`o2open_dispatcher`**.

See Also **`o2server`**, **`o2web_server`**, **`o2web_gateway`**

6 O2Web Reference

o2web_gateway

Summary Starts an O$_2$Web gateway.

Syntax **`o2web_gateway`**

Description This command starts a new O$_2$Web gateway. A gateway is not launched by a user but by an HTTP server in order to answer an OQL query. The **`o2web_gateway`** program complies with the CGI protocol.

Once launched, the O$_2$Web gateway has to find and query the O$_2$Web dispatcher to get the address of an O$_2$Web server that can answer the query. The gateway find the dispatcher host name with the **`O2OPEN_DISPATCHER`** environment variable or in a system-dependent file (**`/etc/o2openaccess`** for UNIX and **`$WINDIR\system32\drivers\etc\o2web`** for Windows NT). The gateway connects to the dispatcher using the TCP port found by querying the system for the port of the **`o2open_dispatcher`** service (Refer to Installation for details of how you can specify this information).

This program is generally installed in a special directory of the HTTP server containing the CGI scripts.

Environment variables The O$_2$Web gateway environment is built by the HTTP server, particularly the CGI environment variables.

`O2OPEN_DISPATCHER`	the dispatcher host name (not with all HTTP servers.

Files **`/etc/o2openaccess`** (UNIX) or **`$WINDIR\system32\drivers\etc\o2web`** (Windows NT)
a file containing the dispatcher host name.

`/etc/services` (UNIX) or **`$WINDIR\system32\drivers\etc\services`** (Windows NT)
a file containing a list of TCP and UDP services.

See Also **`o2open_dispatcher`**, **`o2web_server`**

o2web_server

Summary Starts an O_2Web server.

Syntax **`o2web_server [-v] [-system system_name]`**

`[-server server_host] [-dispatcher dispatcher_host]`

Description This command starts a new O_2Web server on a machine. An O_2Web server is used to answer a query coming from an **`o2web_gateway`**.

When started, **`o2web_server`** establishes a connection with an O_2Web dispatcher (**`o2open_dispatcher`**) which must already be running and establishes also a connection with a named O_2 database system using **`o2server`** which must already be running.

Options

`-v`	display additional information about the **`o2web_server`** activity.
`-system`	specifies the system name. The **`system_name`** must appear in the systems file. If this option is not used, the system name is taken from the environment variable **`O2SYSTEM`** or is given by a directive in the **`.o2rc`** configuration file.
`-server`	specifies the O_2 host server name. If this option is not used, the machine specified by the **`O2SERVER`** environment variable is used. The machine must be on the network. Or the host server name can be specified by a directive in the **`.o2rc`** configuration file.
`-dispatcher`	specifies the O_2Web dispatcher host. If this option is not used, the machine specified by the **`O2OPEN_DISPATCHER`** environment variables is used. If nothing is specified, the machine written in the system-dependent file (**`/etc/o2openaccess`** for UNIX and **`$WINDIR\system32\drivers\etc\o2openaccess`** for Windows NT.

Environment variables The following environment variables are used by this command:

`O2HOME`	Specifies the installation directory of O_2. This variable is mandatory.
`O2OPEN_DISPATCHER`	Indicates the default dispatcher host if the **`-dispatcher`** option is not used.

Files **/etc/o2openaccess** for UNIX and **$WINDIR\system32\drivers\etc\o2openaccess** for Windows NT
a file containing the dispatcher host name.

O2HOME/**.o2serverrc**
a list of system names and associated physical volumes, servers and log files. This file is located in the O_2 installation directory.

See Also **o2server**, **o2open_dispatcher**, **o2web_gateway**

Index

INDEX

INDEX